Technische Universität München

Zentrum Mathematik

# Numerische Untersuchung von Block-Vorkonditionierern zur Lösung der Navier-Stokes-Gleichungen

Diplomarbeit

von

Simon Funke

| | |
|---|---|
| Themensteller: | Prof. Dr. Michael Ulbrich |
| Betreuer: | Prof. Dr. Michael Ulbrich |
| | Dipl.-Math. Florian Lindemann |
| Abgabetermin: | 06.07.2009 |

Ich erkläre hiermit, daß ich die Diplomarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe.

Garching, den

**Zusammenfassung**

Diese Diplomarbeit untersucht verschiedene Lösungsstrategien für die instationären Navier-Stokes-Gleichungen. Die Diskretisierung der Gleichungen erfolgt mit der G2-Methode [9], eine stabilisierte Finite-Elemente-Methode. Ziel ist es, ein effizientes Lösungsverfahren zu entwickeln, dessen Aufwand unabhängig von Parametern wie Viskosität, Gitterfeinheit und Zeitschrittweite ist. Ein weiterer Aspekt bei der Untersuchung ist die Skalier- und Parallelisierbarkeit der Algorithmen.

Der Fokus bei den Lösungsverfahren liegt auf Newton-GMRESR-Kombinationen mit geeigneten Block-Vorkonditionierern für GMRESR. In einem 2D- und 3D-Benchmark werden verschiedene Vorkonditionierer miteinander verglichen. Außerdem treten bei der Anwendung dieser Vorkonditionierer weitere lineare Probleme auf, für die ebenfalls nach geeigneten Lösungsverfahren gesucht wird.

Zu den betrachteten Vorkonditionierern gehört auch eine Erweiterung des LSC-Vorkonditionierers für stabilisierte Diskretisierungen [3]. In dieser Arbeit werden neue Parameter für diesen Vorkonditionierer vorgestellt, die speziell mit der G2-Methode gute Ergebnisse erzielen.

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

Die numerische Simulation turbulenter Strömungen hat zweifellos große Bedeutung in verschiedensten Anwendungsgebieten. Dazu gehören insbesondere die Luft- und Raumfahrttechnik, der Maschinenbau und das Chemie- und Bauingenieurwesen. Als Grundgleichungen dienen die Navier-Stokes-Gleichungen, die sich für die Beschreibung von sowohl laminaren als auch turbulenten Strömungen eignen. Die Simulation turbulenter Strömungen ist eine große Herausforderung für die numerische Mathematik, da die Turbulenzen auf allen Größenskalen auftreten. Um die physikalischen Vorgänge des Flusses vollständig zu repräsentieren, benötigt die numerische Simulation daher eine sehr feine Diskretisierung des Flussgebietes. Dieser Ansatz der direkten numerischen Lösung (DNS) überfordert jedoch schon bei einfachen Problemen die heute verfügbaren Rechnerkapazitäten. Um trotzdem eine Simulation mit vertretbarem Aufwand zu ermöglichen, wird bei der Large Eddy Simulation (LES) zwischen großen und kleinen Wirbelstrukturen unterschieden. Während die Berechnung der großen Wirbel direkt erfolgt, werden kleine Wirbel über ein Turbulenzmodell abgebildet. Ein solches Turbulenzmodell verwendet auch die von J. Hoffman und C. Johnson entwickelte $G2$-Methode [7, 9] zur Lösung der Navier-Stokes-Gleichungen, die in dieser Arbeit als Basis dient.

Aber auch mit dem Turbulenzmodell der $G2$-Methode müssen während des Lösungsvorgangs nichtlineare Probleme mit $10^5$ bis $10^7$ Unbekannten gelöst werden. Ziel der Diplomarbeit ist es, schnelle Lösungsverfahren für genau diese Probleme zu finden. Dafür werden unterschiedliche Ansätze miteinander verglichen. Sie alle haben gemeinsam, dass das nichtlineare Problem zunächst linearisiert wird, um die Lösung iterativ zu bestimmen. In jedem solchen Iterationsschritt müssen ein oder mehrere lineare Probleme gelöst werden, die ähnlich viele Unbekannte wie das Ursprungssystem haben.

Neben der Wahl des nichtlinearen Verfahrens stellt sich also die Frage nach geeigneten linearen Lösungsmethoden. Der Rechen- und Speicheraufwand von direkten Methoden wie die LU-Zerlegung verhindert deren Einsatz bei großen Problemdimensionen. Stattdessen werden auch hier iterative Lösungsverfahren verwendet. Im Rahmen dieser Arbeit stehen dafür Krylov-Unterraum-Verfahren im Vordergrund. Ein zufriedenstellendes Ergebnisse erhält man mit diesen Verfahren aber erst durch

Kombination mit einem geeigneten Vorkonditionierer. Ein (rechter) Vorkonditionierer erweitert ein lineares System $Ax = b$ durch $AM^{-1}y = b$. Die Lösung $x$ erhält man dann mit $M^{-1}y = x$. Das Inverse des Vorkonditionierers $M$ sollte einerseits einfach zu konstruieren und anzuwenden sein, und andererseits zu einer schnellen Konvergenz des Krylov-Unterraum-Verfahrens bei der Bestimmung von $y$ führen. Die Wahl des Vorkonditionierers hat einen wesentlichen Einfluss auf die Effizienz des Lösungsverfahrens. Der Hauptteil der Diplomarbeit konzentriert sich daher auf die Bestimmung guter Vorkonditionierer.

Der zweite Teil der Arbeit betrachtet die Anwendung dieser Vorkonditionierer. Es treten wiederum lineare Probleme auf, für die ebenfalls schnelle Verfahren benötigt werden. Hier bieten sich Mehrgitterverfahren an, die genauer analysiert werden.

Es folgt nun eine Einführung der Navier-Stokes-Gleichungen sowie deren Diskretisierung und Stabilisierung mit der $G2$-Methode.

## 1.1 Die Navier-Stokes-Gleichungen

Ausgangspunkt ist die instationäre Form der inkompressiblen Navier-Stokes-Gleichungen in einem geschlossenen Gebiet $\Omega \subset \mathbb{R}^n$, $n = 2, 3$ und einem Zeitintervall $I = [0, T]$:

$$u_t + (u \cdot \nabla)u - \nu\Delta u + \nabla p = f \qquad \text{in } \Omega \times I \tag{1.1}$$

$$\nabla \cdot u = 0 \qquad \text{in } \Omega \times I \tag{1.2}$$

Dabei ist $\nu \in \mathbb{R}$ die kinematische Viskosität. Der Quellterm $f : \Omega \times I \to \mathbb{R}^n$ dient zur Beschreibung von Volumenkräften wie z.B. Gravitation. Die gesuchten Funktionen $u = (u_i)_{i=1..n} : \Omega \times I \mapsto \mathbb{R}^n$ und $p : \Omega \times I \mapsto \mathbb{R}$ entsprechen dem Geschwindigkeits- und Druckfeld des Fluids. Der Kern der Navier-Stokes-Gleichungen ist der Impulssatz. Zusammen mit der Energieerhaltung bildet er die Impulsgleichung 1.1. Die Kontinuitätsgleichung 1.2 entsteht aus der Massenerhaltung.

Um ein Strömungsproblem vollständig zu beschreiben, werden neben 1.1 und 1.2 zusätzliche Anfangswerte und Randbedingungen benötigt. Der Anfangswert $u^0$ gibt das Geschwindigkeitsfeld zum Zeitpunkt $t = 0$ vor:

$$u(\cdot, 0) = u^0 \qquad \text{in } \Omega$$

Bei den Randbedingungen unterscheidet man typischerweise zwischen Dirichlet- und Neumann-Randbedingung. Die Dirichlet-Randbedingung fixiert die Fluidgeschwindigkeit auf dem Rand $\Gamma_D \subset \partial\Omega$, während eine Neumann-Randbedingung die Normalableitung der Lösung auf $\Gamma_N \subset \Omega$ vorgibt. Mit $\partial\Omega = \Gamma_D \cup \Gamma_N$, $\Gamma_D \cap \Gamma_N = \emptyset$ und $w : \Gamma_D \to \mathbb{R}^n$ und $s : \Gamma_N \to \mathbb{R}^n$ erhält man also die Bedingungen

$$u(., t) = w \qquad \text{auf } \Gamma_D$$

$$\nu \frac{\partial u(., t)}{\partial n} - p(., t) \cdot n = s \qquad \text{auf } \Gamma_N$$

für alle $t \in I$. Dabei ist $n$ der nach außen gerichtete Normalenvektor von $\Omega$. Ist die Dirichlet-Randbedingung $w = 0$, das heißt das Fluid haftet am Rand, heißt sie auch "no-slip"-Randbedingung. Die Neumann-Randbedingung $s = 0$ ist als natürliche Randbedingung bekannt. Physikalisch gesehen verhält sich das Fluid dort wie an einem angrenzenden, unendlich großen Behälter.

*Bemerkung.* Falls reine Dirichlet-Randbedingungen gefordert werden, das heißt falls $\Gamma_D = \partial\Omega$, muss eine zusätzliche Randbedingung den Druck an einem Punkt fixieren. Der Grund ist, dass in diesem Fall keine Kopplung von Druck- und Geschwindigkeitsfeld durch die Randbedingungen gegeben ist und in den Navier-Stokes-Gleichungen nur die Ableitung von $p$ vorkommt. Damit wäre $p$ nur bis auf eine Konstante definiert. Zudem muss in diesem Fall die Funktion $w$ einer Kompatibilitätsbedingung genügen (siehe z.B. [4, S. 215]). Der Einfachheit halber gehen wir deshalb im Folgenden von $\Gamma_D \neq \partial\Omega$ aus.

## 1.2 Finite-Elemente-Methode

Zur numerischen Lösung der Navier-Stokes-Gleichungen müssen diese diskretisiert werden. Ein gängiges Verfahren ist die Finite-Elemente-Methode, welche in diesem Abschnitt vorgestellt wird. Die Diskretisierung erfolgt in drei Schritten:

1. Bildung der schwachen Form.
2. Diskretisierung im Ort.
3. Diskretisierung in der Zeit.

Das Ergebnis ist ein nichtlineares endlich-dimensionales Gleichungssystem, das es zu lösen gilt. Um die folgenden Herleitungen einfacher zu gestalten, wird angenommen, dass auf den Neumann-Rand die natürliche Randbedingung gilt, also $s = 0$ auf $\Gamma_N$.

### 1.2.1 Bildung der schwachen Form

Anstatt klasssische Lösungen der Navier-Stokes-Gleichungen zu suchen, werden wir die schwache Form der Gleichungen 1.1 und 1.2 bilden und eine Lösung aus dem Raum der schwach differenzierbaren Funktionen berechnen. Um die schwache Form definieren zu können, werden zunächst geeignete Lösungs- und Testfunktionenräume benötigt. Diese sind der $L^2(\Omega)$ und (affine) Teilräume des Sobolew-Raums $\mathcal{X} = \mathcal{H}^1(\Omega)^n$ und definiert als:

$$\begin{aligned} \mathcal{X}_w &:= \{u \in \mathcal{X} \,|\, u = w \quad \text{auf } \Gamma_D\} \\ \mathcal{X}_0 &:= \{v \in \mathcal{X} \,|\, v = 0 \quad \text{auf } \Gamma_D\} \\ \mathcal{M} &:= L^2(\Omega) \end{aligned}$$

Der Lösungsraum für das Geschwindigkeitsfeld $\mathcal{X}_w$ stellt automatisch die Einhaltung der Dirichlet-Randbedingung sicher. Da die instationären Navier-Stokes-Gleichungen

betrachtet werden, hängt die gesuchte Lösung $(u, p)$ zudem noch von der Zeit ab. Tatsächlich muss $u$ sogar nach $t$ differenzierbar sein. Wir brauchen also die Erweiterung auf die Bochner-Räume:

$$\begin{aligned}
\bar{\mathcal{X}} &:= \{u \in L^2(I, \mathcal{X}) \mid \ u_t \in L^2(I, \mathcal{X}')\} \\
\bar{\mathcal{X}}_w &:= \{u \in \bar{\mathcal{X}} \mid u(\cdot, t) \in \mathcal{X}_w \quad \text{für fast alle } t \in I\} \\
\bar{\mathcal{X}}_0 &:= \{u \in \bar{\mathcal{X}} \mid u(\cdot, t) \in \mathcal{X}_0 \quad \text{für fast alle } t \in I\} \\
\bar{\mathcal{M}} &:= L^2(I, \mathcal{M})
\end{aligned}$$

Dabei bezeichnet $\mathcal{X}'$ den Dualraum von $\mathcal{X}$. Nun kann die schwache Form der Navier-Stokes-Gleichungen definiert werden. Dafür werden die Gleichungen 1.1 bzw. 1.2 zu einem festen Zeitpunkt $t$ mit Testfunktionen $v \in \mathcal{X}_0$ bzw. $q \in \mathcal{M}$ multipliziert und anschliessend über $\Omega$ integriert. Mit der Notation

$$(x, y) := \int_\Omega x(t) \cdot y \qquad \text{für } x \in \bar{\mathcal{X}} \cup \bar{\mathcal{M}},\ y \in \mathcal{X} \cup \mathcal{M}$$

und

$$(u_t, v) := \int_\Omega u_t(t)(v) \qquad \text{für } u \in \bar{\mathcal{X}},\ v \in \mathcal{X}$$

erhält man:

Finde $(u, p) \in \bar{\mathcal{X}}_w \times \bar{\mathcal{M}}$, so dass für fast alle $t \in I$ gilt:

$$\begin{aligned}
(u_t, v) + (u \cdot \nabla u, v) - \nu(\Delta u, v) + (\nabla p, v) &= (f, v) && \forall v \in \mathcal{X}_0 \\
(\nabla \cdot u, q) &= 0 && \forall q \in \mathcal{M}
\end{aligned}$$

Die Anwendung des Gauß'schen Satzes und einer Greenschen Formel liefert:

$$-\nu(\Delta u, v) + (\nabla p, v) = \nu(\nabla u : \nabla v) - (p, \nabla \cdot v) - \int_{\partial\Omega} (\nu \frac{\partial u}{\partial n} - pn) \cdot v \tag{1.3}$$

Dabei ist $\nabla u : \nabla v$ das Produkt zweier Tensoren 2. Stufe und definiert als:

$$(\nabla u : \nabla v) := \int_\Omega \sum_{i,j=1}^{n} \frac{\partial u_i(t)}{\partial x_j} \cdot \frac{\partial v_i}{\partial x_j}$$

Wegen $v = 0$ auf $\Gamma_D$ enthält das Randintegral in 1.3 gerade die Neumann-Randbedingung. Mit der Annahme $s = 0$ fällt dieses Randintegral weg und die schwache Form lautet schließlich:

Finde $(u, p) \in \bar{\mathcal{X}}_w \times \bar{\mathcal{M}}$, so dass für fast alle $t \in I$ gilt:

$$\begin{aligned}
(u_t, v) + (u \cdot \nabla u, v) + \nu(\nabla u : \nabla v) - (p, \nabla \cdot v) &= (f, v) && \forall v \in \mathcal{X}_0 && (1.4) \\
(\nabla \cdot u, q) &= 0 && \forall q \in \mathcal{M} && (1.5)
\end{aligned}$$

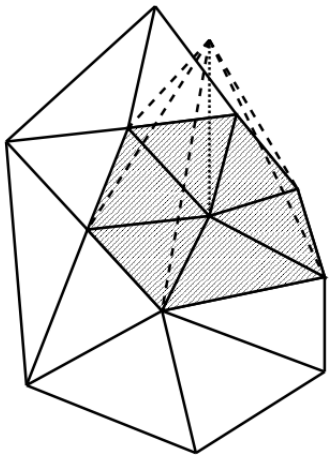

Abbildung 1.1: Eine P1 Basisfunktion auf dem triangularisiertem Gebiet. (Quelle: [4])

### 1.2.2 Diskretisierung im Ort

Die Diskretisierung der schwachen Form im Ort erfolgt mithilfe der Galerkin-Projektion. Die Idee ist, die Lösungs- und Testfunktionenräume im Ort auf endliche Teilräume einzuschränken. Seien also $X^h \subset \mathcal{X}$ und $M^h \subset \mathcal{M}$ diese endlichen Teilräume. Analog wie im vorherigen Abschnitt seien $X_0^h$ und $X_w^h$ die Menge der Funktionen aus $X^h$, die die entsprechenden Dirichlet-Randbedingung erfüllen. Das semidiskrete Analogon zu 1.4 und 1.5 ist dann:

Finde $(u, p) \in \bar{\mathcal{X}}_w^h \times \bar{\mathcal{M}}^h$, so dass für jeden Zeitpunkt $u(t) \in X_w^h$ und $p(t) \in M^h$ gilt und:

$$(u_t, v) + (u \cdot \nabla u, v) + \nu(\nabla u : \nabla v) - (p, \nabla \cdot v) = (f, v) \qquad \forall v \in X_0^h \qquad (1.6)$$

$$(\nabla \cdot u, q) = 0 \qquad \forall q \in M^h \qquad (1.7)$$

Inwieweit die numerische Lösung eine gute Approximation an die analytische Lösung ist, hängt stark von der Wahl der Räume $X^h$ und $M^h$ ab. Für einfache partielle Differentialgleichungen zeigt das Lemma von Céa [1], dass die Genauigkeit der numerischen Lösung im wesentlichen davon abhängt, wie gut die analytische Lösung im endlichen Funktionenraum dargestellt werden kann. Zu den bekanntesten dieser geeigneten Räume gehören die Finite-Elemente-Räume. Sie nutzen die Tatsache, dass glatte Funktionen beliebig gut mit stückweisen Polynomen approximiert werden können. Hierzu wird das Rechengebiet $\Omega$ in ein Gitter von Polyedern (z.B. Dreiecke in 2D oder Tetraeder in 3D) mit Durchmesser $h$ partitioniert. Für jeden Knoten $j$ in diesem Gitter ist nun eine Basisfunktion definiert, die ihren Träger nur auf den Polyedern mit Knoten $j$ als Eckpunkt hat. Abbildung 1.1 illustriert eine lineare Finite-Elemente-Basisfunktion in einem triangulisierten Gebiet. Wegen des linearen Ansatzes werden diese Finite-Elemente auch als P1-Elemente bezeichnet. Analog sind P0-Elemente stückweise konstant und P2-Elemente stückweise quadratisch.

Ein weiterer wünschenswerter Punkt bei der Wahl der Räume $X^h$ und $M^h$ ist, dass sie die Inf-Sup-Bedingung erfüllen: Es gibt eine von der Gitterweite $h$ unabhängige Konstante $\gamma > 0$, so dass gilt:

$$\min_{q \in M^h, q \neq constant} \max_{v \in X^h, c \neq 0} \frac{|(q, \nabla v)|}{||v||_{\mathcal{X}} ||q||_{\mathcal{M}}} \geq \gamma$$

Sie garantiert, dass 1.6 und 1.7 unabhängig vom gewählten Gitter gelöst werden kann. Erfüllen $X^h$ und $M^h$ die Inf-Sup-Bedingung, so heißen sie stabil, ansonsten instabil. Leider sind viele scheinbar natürlichen Kombinationen von Finite-Elemente-Räumen instabil. Dazu gehören insbesondere die "low-order" Kombinationen wie $P_1 - P_0$, also stückweise lineare Elemente für $X^h$ und konstante Elemente für $M^h$. Aber auch "equal-order" Kombinationen, wie $P_1 - P_1$ erfüllen die Inf-Sup-Bedingung nicht. Die Verwendung solcher Finite-Elemente-Räume ist nur mit Stabilisierungstechniken möglich, auf die später noch eingegangen wird.

### 1.2.3 Diskretisierung der Zeit

Das Ergebnis der Galerkin-Projektion im Ort ist eine differentiell-algebraische Gleichung bezüglich der Zeit. Für deren Lösung kann prinzipiell jedes Diskretisierungsverfahren für Anfangswertprobleme aus dem Bereich der gewöhnlichen Differentialgleichung verwendet werden. Bei der Wahl muss allerdings darauf geachtet werden, dass die Gleichung 1.6 wegen des Diffusionsterms $\nu(\nabla u : \nabla v)$ steif ist und deshalb bei nicht A-stabilen Verfahren eine Schrittweitenbeschränkung erzwingt. Eine gute Übersicht verschiedener Methoden ist in [6, S. 325ff] zu finden.

Zur Vertreter der A-stabilen Verfahren gehören die cG(q) bzw. dG(q) Verfahren. Sie basieren auf der Idee, die Zeit analog wie den Ort zu diskretisieren. Bei der Bildung der schwachen Form in Abschnitt 1.2.1 wird dann nicht nur über den Ort, sondern auch über einen Zeitschritt integriert. Sei dafür $I = [0, T] = \{0\} \cup I_1 \cup I_2 \cup ... \cup I_N$ eine Partitionierung von I in Zeitintervalle $I_n = (t_{n-1}, t_n]$ der Länge $k_n = t_n - t_{n-1}$. Die endlichen Teilräume $X^h$ und $M^h$ werden nun analog zu vorher zu endlichen Teilräumen $\bar{X}^h \subset \bar{\mathcal{X}}$ und $\bar{M}^h \subset \bar{\mathcal{M}}$ erweitert. $\bar{X}_0^h$ und $\bar{X}_w^h$ seien die affinen Teilräume von $\bar{X}^h$, die die entsprechenden Dirichlet-Randbedingungen erfüllen. Zeitintegration der semidiskretisierten Form 1.6 und 1.7 über das Zeitintervall $I_n$ ergibt dann:

Finde $(u, p) \in \bar{X}_w^h \times \bar{M}^h$, so dass für jedes Zeitintervall gilt:

$$\begin{aligned} \int_{t_{n-1}}^{t_n} (u_t, v) + (u \cdot \nabla u, v) + \nu(\nabla u : \nabla v) - (p, \nabla \cdot v)\, dt & \\ = \int_{t_{n-1}}^{t_n} (f, v)\, dt \qquad \forall v \in \bar{X}_0^h & \qquad (1.8) \\ \int_{t_{n-1}}^{t_n} (\nabla \cdot u, q)\, dt = 0 \qquad \forall q \in \bar{M}^h & \end{aligned}$$

Ein natürlicher Ansatz ist es, $\bar{X}^h$ und $\bar{M}^h$ so zu wählen, dass die Funktionen $u$ und $p$ bezüglich der Zeit stückweise Polynome vom Grad q sind. Pro Zeitintervall werden

dann $q+1$ Bedingungen benötigt, um die Lösung eindeutig festzulegen. Mit der Forderung einer stetigen Lösung auf ganz $I$ ist $u$ und $p$ am Zeitpunkt $t_{n-1}$ vorgegeben. Weitere $q$ Bedingungen kommen hinzu, wenn die Testfunktionen $v$ und $q$ in der Zeit Polynome vom Grad $q-1$ sind. Dieser Ansatz heißt stetige Galerkin-Methode oder cG(q)-Methode, weil die Lösungsfunktionen stetig in der Zeit sind. Verzichtet man auf die Stetigkeit der Lösung zwischen den Zeitintervallen, erhält man die diskontinuierliche Galerkin-Methode, kurz dG(q). Bei Verwendung von Finite-Elementen vom Grad p im Ort für $X^h$ und $M^h$ und Polynome vom Grad q in der Zeit spricht man insgesamt von einem cG(p)cG(q)-Verfahren.

Betrachten wir nun den Spezialfall $q = 1$. Die Testfunktionen sind dann innerhalb eines Zeitintervalls konstant. Wählt man als Quadraturformel für die Zeitintegrale in 1.8 die Mittelpunktsregel, so berechnet sich die Lösung $(u^n, p^n) \in X_w^h \times M^h$ zum Zeitschritt $t_n$ aus der Lösung des alten Zeitschritts $(u^{n-1}, p^{n-1})$ mittels

$$\begin{aligned}(u^n - u^{n-1}, v) + k_n(\bar{u}^n \cdot \nabla \bar{u}^n, v) + k_n \nu(\nabla \bar{u}^n : \nabla v) - k_n(p^n, \nabla \cdot v) & \\ = k_n(f, v) \qquad \forall v \in X_0^h & \qquad (1.9)\\ (\nabla \cdot \bar{u}^n, q) = 0 \qquad \forall q \in M^h &\end{aligned}$$

wobei $\bar{u} = \frac{1}{2}(u^n + u^{n-1})$ ist. Die Schreibweise $(\cdot, \cdot)$ und $(\cdot : \cdot)$ ist wie vorher definiert, nur dass die Argumente hier nicht zeitabhängig sind. Man beachte, dass der Druckterm in 1.9 rein implizit aufgelöst, da keine Anfangsbedingungen für den Druck gegeben sind.

Im weiteren wird eine Matrixdarstellung von 1.9 und die Ankopplung der Randbedingungen hergeleitet. Sie macht deutlich, wo genau die Nichtlinearität des Systems enthalten ist und wird später für die Herleitung von Lösungsverfahren verwendet. Sei dafür $\{\Phi_i\}_{i=1..m_u}$ die Finite-Elemente-Basis von $X_0^h$ und $\{\Phi_i\}_{i=m_u+1,\ldots,m_\partial}$ die Erweiterung zum Erzeugnis von $X_w^h$. Die diskrete Geschwindigkeitsfunktion zum Zeitpunkt $t_n$ kann dann geschrieben werden als:

$$u^n = \sum_{j=1}^{m_u} \mathbf{u}_j^n \Phi_j + \sum_{j=m_u+1}^{m_\partial} \mathbf{u}_j^n \Phi_j$$

Die Koeffizienten $\mathbf{u}_j^n$, $j = m_u + 1, .., m_\partial$ sind durch die Dirichlet-Randbedingungen auf $\Gamma_D$ vorgegeben. Analog kann die diskrete Druckfunktion mithilfe der Finite-Elemente-Basis $\{\Psi_i\}_{i=1..m_p}$ von $M^h$ geschrieben werden als:

$$p^n = \sum_{j=1}^{m_p} \mathbf{p}_j^n \Psi_j$$

Einsetzen dieser Darstellungen in 1.9 und anschließendes Testen mit den Finite-Elemente-Basisfunktionen von $X_0^h$ und $M^h$ führt auf eine (nichtlineare) Matrixdarstellung von 1.9 der Form

$$\begin{pmatrix} F(u^n) & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \qquad (1.10)$$

mit

$$\begin{aligned} F(u^n)_{ij} &= 2k_n^{-1}(\Phi_j, \Phi_i) + (\bar{u}^n \cdot \nabla\Phi_j, \Phi_i) + \nu(\nabla\Phi_j : \nabla\Phi_i) \\ B_{ik}^T &= -2(\Psi_k, \nabla\Phi_i) \\ B_{kj} &= (\nabla\Phi_j, \Psi_k) \end{aligned}$$

für $i = 1...m_u$, $j = 1...m_\partial$ und $k = 1...m_p$. Die rechte Seite $(\mathbf{r}_u, \mathbf{r}_p)^T$ enthält die Werte des letzten Zeitschritts und des Koeffizientenvektors $\mathbf{f}$ von f. Man beachte, dass die Nichtlinearität des Systems in dem $F(u^n)$-Block steckt: Der Block selber hängt linear von $u^n$ ab und wird in 1.10 mit dem Koeffizientenvektor $\mathbf{u}^n$ multipliziert. Damit ist die Gleichung insgesamt quadratisch in $u^n$.

In der Matrixdarstellung 1.10 sind jedoch die Dirichlet-Randbedingungen noch nicht berücksichtigt. Dies gelingt durch Hinzufügen einer Identitätsfunktion auf dem Dirichlet-Rand in Form einer $(m_\partial - m_u) \times m_\partial$ Blockmatrix $D$ und einer rechten Seite $\mathbf{r}_\partial$ mit den Werten der Dirichlet-Randbedingung:

$$\begin{pmatrix} F(u^n) & B^T \\ D & 0 \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_\partial \\ \mathbf{r}_p \end{pmatrix}$$

Um die Darstellung einfach zu halten, werden im Folgenden die ersten beiden Zeilen zusammengefasst und die Darstellung 1.10 weiterhin verwendet. Man beachte, dass mit dieser Konvention $F(u^n)$ eine quadratische $m_\partial \times m_\partial$ Matrix ist.

*Bemerkung.* Die (natürliche) Neumann-Randbedingung ist schon bei der Bildung der schwachen Form berücksichtigt worden, siehe Abschnitt 1.2.1.

## 1.3 Das stabilisierte cG(1)cG(1)-Verfahren

Betrachten wir nochmals die Ordnungen der Finite-Elemente-Funktionen beim cG(1)cG(1)-Verfahren: Im Druck- und Geschwindigkeitsraum werden stetige Funktionen verwendet, die stückweise linear in Ort und Zeit sind. Die zugehörigen Testfunktionen sind ebenfalls linear im Ort, aber konstant in der Zeit. Durch die Verwendung derselben Polynomordnungen für die Druck- und Geschwindigkeitsfunktionen ist das Verfahren in der Praxis sehr attraktiv: Die konsistente Wahl vereinfacht die notwendigen Datenstrukturen und die einfache Form der Basiselemente ermöglicht schnelle numerische Berechnungen. Allerdings kann das cG(1)cG(1)-Verfahren ohne eine besondere Stabilisierungstechnik nicht erfolgreich auf die instationären Navier-Stokes-Gleichungen angewandt werden. Das hat im wesentlichen zwei Gründe:

Wie schon erwähnt, verwendet cG(1)cG(1) sowohl im Geschwindigkeits- als auch im Druckraum lineare Finite-Elemente im Ort. Diese "equal-order" Kombination erfüllt die Inf-Sup-Bedingung nicht. Daher würden ohne spezielle Stabilisierung Oszillationen im Druckraum auftreten, die auch als "checkerboard mode" bekannt sind.

Die zweite Instabilität tritt auch bei anderen Finite-Elemente-Räumen auf. Während das Lösen von diffusionsdominanten Strömungen mit der Standard-Galerkin-Projektion zufriedenstellend funktioniert, stellt man bei konvektionsdominanten Problemen Störungen in Gestalt von Oszillationen im Geschwindigkeitsfeld fest. Diese Störungen treten auf, sobald das Gitter nicht fein genug ist, um scharfe Übergänge zwischen zwei Strömungsschichten aufzulösen. Von dort aus breiten sie sich auch auf benachbarte Gebiete aus. Eine Abhilfe gegen diese Instabilität wäre eine gezielte Verfeinerung des Gitters an den Problemstellen, wo Grenzschichten genauer aufgelöst werden müssen. Bei komplexen Problemen, in denen die Gitterverfeinerung nicht manuell oder im vornherein durchgeführt werden kann, wird dafür eine adaptive Gittersteuerung verwendet: Zuerst wird auf einem groben Startgitter die diskrete Lösung berechnet. Anschließend verfeinert man mit Hilfe eines a posteriori Fehlerschätzers das Gitter dort, wo der geschätzte Fehler groß ist. Um eine adaptive Gittersteuerung effektiv einsetzen zu können, ist es deshalb wichtig, dass sich Diskretisierungsfehler nicht weiter ausbreiten. Insbesondere ist daher die Standard-Galerkin-Projektion bei konvektionsdominanten Strömungen nicht für eine adaptive Gittersteuerung geeignet.

Beide Störungen können mit einer geeigneten Stabilisierung behandelt werden. Zu den bekanntesten Stabilisierungstechniken gehören die Streamline-Upwind/Petrov-Galerkin Methode (SUPG) [2] und die Galerkin/Least-Squares Methode (GLS) [22, 28]. Eine auf der GLS-Methode aufbauende Stabilisierung ist die "General Galerkin" Methode, kurz G2-Methode von J. Hoffmann und C. Johnson [7, 9]. Sie liefert erfolgversprechende Ergebnisse in mehreren Anwendungen [8, 11] und soll daher als Grundlage für die Stabilisierung dienen.

Die Herleitung der G2-Methode für das cG(1)cG(1)-Verfahren wird mit der alternativen Formulierung einer Petrov-Galerkin Methode durchgeführt. Bei dieser werden die Testfunktionen, die bei der Bildung der schwachen Form verwendet werden, mit Stabilisierungstermen versehen. Konkret wird die Testfunktion $v$ ersetzt durch

$$v + \delta_1(v_t + u \cdot \nabla v - \nu\Delta v + \nabla q)$$

und $q$ durch

$$q + \delta_2 \nabla \cdot v$$

wobei $\delta_1$ und $\delta_2$ Stabilisierungsparameter sind. Dabei müssen die Finite-Elemente-Räume so definiert sein, dass $v_t$, $\Delta v$ und $\nabla q$ existieren. Die stabilisierte schwache Form erhält man analog zu Abschnitt 1.2.1, jedoch unter Verwendung der stabilisierten Testfunktionen. Sie lautet

$$\begin{aligned}(u_t, v) + (u \cdot \nabla u, v) + \nu(\Delta u, v) - (\nabla p, v) + SD_\delta(u, p; v, 0) &= (f, v)\\ (\nabla \cdot u, q) + SD_\delta(u, p; 0, q) &= 0\end{aligned}$$

mit dem Stabilisierungsterm:

$$SD_\delta(u,p;v,q) := \delta_1(u\cdot\nabla u - \nu\Delta u + \nabla p - f, v_t + u\cdot\nabla v - \nu\Delta v + \nabla q) \\ +\delta_2(\nabla\cdot v, \nabla\cdot u)$$

*Bemerkung.* Um die Zeitdiskretisierung einfach zu halten, wurde die Stabilisierung nicht im $u_t$ Term durchgeführt, siehe [11, S. 216]

Bei einer cG(1)cG(1)-Diskretisierung sind die Testfunktionen linear im Ort und konstant in der Zeit und die Lösungsfunktion linear im Ort. Daher vereinfacht sich der Stabilisierungsterm zu:

$$SD_\delta(u,p;v,q) = \delta_1(u\cdot\nabla u + \nabla p - f, u\cdot\nabla v + \nabla q) + \delta_2(\nabla\cdot u, \nabla\cdot v)$$

Nun führt man die Diskretisierung analog zu Abschnitt 1.2.2 und 1.2.3 durch und erhält die stabilisierte diskrete Form:

Für $n = 1...N$ finde $(u^n, p^n) \in X_w^h \times M^h$, so dass

$$\begin{aligned}(u^n - u^{n-1}, v)k_n^{-1} + (\bar{u}^n\cdot\nabla\bar{u}^n, v) + \nu(\nabla\bar{u}^n : \nabla v) - (p^n, \nabla\cdot v) \\ +SD_\delta(\bar{u}^n, p^n; v, 0) = (f, v) \qquad \forall v \in X_0^h \\ (\nabla\cdot\bar{u}^n, q) + SD_\delta(\bar{u}^n, p^n; 0, q) = 0 \qquad \forall q \in M^h\end{aligned} \tag{1.11}$$

mit $\bar{u}^n := \frac{1}{2}(u^n + u^{n-1})$ gilt.

Bleibt noch die Wahl der Parameter $\delta_1$ und $\delta_2$ im Stabilisierungsterm. In der G2-Methode werden diese folgendermaßen gesetzt:

$$\delta_1 = \begin{cases} \frac{1}{2}(k_n^{-2} + |u^{n-1}|^2 h^{-2})^{-\frac{1}{2}}, & \text{falls } \nu < u^{n-1}h \\ \kappa_1 h^2, & \text{sonst}\end{cases}$$

$$\delta_2 = \begin{cases} \kappa_2 h, & \text{falls } \nu < u^{n-1}h \\ \kappa_2 h^2, & \text{sonst}\end{cases}$$

Dabei sind $\kappa_1, \kappa_2$ zwei positive Konstanten und $h$ die Elementgröße des Gitters. Die Fallunterscheidung bei der Definition von $\delta_1$ und $\delta_2$ ermöglicht es, konvektionsdominante Bereiche der Strömung ($\nu < u^{n-1}h$) anders zu gewichten als diffusionsdominante und fungiert damit als einfaches automatisches Turbulenzmodell.

Analog zu Abschnitt 1.2.3 lässt sich die Matrixdarstellung erstellen. Sie lautet

$$\begin{pmatrix} F(u^n) & B^T(u^n) \\ B(u^n) & -C \end{pmatrix} \begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{1.12}$$

mit:

$$\begin{aligned} F(u^n)_{ij} =& 2k_n^{-1}(\Phi_j, \Phi_i) + (\bar{u}^n\cdot\nabla\Phi_j, \Phi_i) + \nu(\nabla\Phi_j : \nabla\Phi_i) \\ &+ \delta_1(\bar{u}^n\cdot\nabla\Phi_j, \bar{u}^n\cdot\nabla\Phi_i) + \delta_2(\nabla\cdot\Phi_j, \nabla\cdot\Phi_i) \\ B^T(u^n)_{ik} =& -2(\Psi_k, \nabla\Phi_i) + 2\delta_1(\nabla\Psi_k, \bar{u}^n\cdot\nabla\Phi_i) \\ B(u^n)_{kj} =& (\nabla\Phi_j, \Psi_k) + \delta_1(\bar{u}^n\cdot\nabla\Phi_j, \nabla\Psi_k) \\ C_{kl} =& -\delta_1(\nabla\Psi_k, \nabla\Psi_l) \end{aligned} \tag{1.13}$$

Im Vergleich zum unstabilisierten Fall 1.10 werden die Blockmatrizen $F$, $B^T$ und $B$ durch die Stabilisierungsterme erweitert und hängen nun von $u^n$ ab. Damit sind die Nichtlinearitäten des Systems in diesen drei Blöcken enthalten. Auch der Nullblock verschwindet und wird durch die Stabilisierungsmatrix $-C$ ersetzt.

Betrachten wir zuletzt noch den Aufbau der Blockmatrizen. Die Matrix $F$ ist die Summe einer Massenmatrix, dem diskreten Analogon des Konvektionssterm mit Geschwindigkeitfeld $u^n$, einem Laplace-Operator und schließlich den Stabiliserungstermen. $B^T$ entspricht einem stabilisierten Gradientenoperator und $B$ ist ein, ebenfalls stabilisierter, diskreter Divergenzoperator. Obwohl durch die Stabilisierungsterme die Symmetrie von $(BB^T)$ verloren geht, wird im Folgenden die Notation beibehalten. Man beachte, dass in der Matrixdarstellung die Dirichlet-Randbedingung wie am Ende von Abschnitt 1.2.3 berücksichtigt werden muss.

# Kapitel 2

# Lösungsverfahren

Im letzten Kapitel wurden die Navier-Stokes-Gleichungen mit dem stabilisierten cG(1)cG(1)-Verfahren diskretisiert, was zu dem Gleichungssystem

$$\begin{pmatrix} F(u^n) & B^T(u^n) \\ B(u^n) & -C \end{pmatrix} \begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{2.1}$$

führte. In diesem Kapitel geht es um Lösungsverfahren, die geeignet sind, dieses System zu lösen. Die Nichtlinearität des Konvektionsterms und der Stabilisierungsterme in 2.1 machen es notwendig, nichtlineare Lösungsverfahren zur Berechnung von $(u^n, p^n)$ einzusetzen. Drei Verfahren werden in diesem Kapitel eingeführt und auf die diskretisierten Navier-Stokes-Gleichungen angewendet: das Newton-Verfahren, die Picard-Iteration und ein Splitting-Verfahren. Alle drei gehören zur Klasse der Iterationsverfahren, das heißt aus Initialwerten $(u^{n,0}, p^{n,0})$ wird sukzessive eine Sequenz von Iterationen $(u^{n,1}, p^{n,1})$, $(u^{n,2}, p^{n,2})$ , $(u^{n,3}, p^{n,3})$, ... berechnet. Konvergiert diese Folge, wird mit einem geeigneten Abbruchkriterium das Iterationsverfahren gestoppt und die letzte Iteration als Lösung ausgegeben.

## 2.1 Newton-Verfahren

Das Newton-Verfahren ist das Standardverfahren zur numerischen Lösung nichtlinearer Gleichungssysteme und ist daher ein naheliegender Ansatz für die Lösung der diskretisierten Navier-Stokes-Gleichungen.

Zur Anwendung des Newton-Verfahrens wird das nichtlineare Gleichungssystem zunächst in ein Nullstellenproblem $H(u^n, p^n) = 0$ umformuliert. Die grundlegende Idee ist nun, die Funktion $H$ an der aktuellen Iteration zu linearisieren und die Nullstelle dieser linearisierten Gleichung als verbesserte Näherung zu verwenden. Mit der Jacobi-Matrix $J_H$ von $H$ erhält man die Formel:

$$J_H(u^{n,i}, p^{n,i})(\delta u^{n,i}, \delta p^{n,i}) = -H(u^{n,i}, p^{n,i})$$

Die Newton-Korrektur $u^{n,i+1} := u^{n,i} + \delta u^{n,i}$ und $p^{n,i+1} := p^{n,i} + \delta p^{n,i}$ liefert dann die nächste Iteration.

Wir werden nun das Newton-Verfahren auf die Variationsformulierung 1.11 der diskretisierten Navier-Stokes-Gleichungen anwenden. Mit $\bar{u}^{n,i} := \frac{1}{2}(u^{n,i}+u^{n-1})$ lautet die linearisierte Impulsgleichung:

$$\begin{aligned}(\delta u^{n,i}, v)k_n^{-1} + (\frac{\delta u^{n,i}}{2} \cdot \nabla \bar{u}^{n,i}, v) + (\bar{u}^{n,i} \cdot \nabla \frac{\delta u^{n,i}}{2}, v) \\ +\nu(\nabla \frac{\delta u^{n,i}}{2} : \nabla v) - (\delta p^{n,i}, \nabla \cdot v) \\ +\delta_1(\bar{u}^{n,i} \cdot \nabla \bar{u}^{n,i} + \nabla p^{n,i} - f, \frac{\delta u^{n,i}}{2} \cdot \nabla v) \\ +\delta_1(\frac{\delta u^{n,i}}{2} \cdot \nabla \bar{u}^{n,i} + \bar{u}^{n,i} \cdot \nabla \frac{\delta u^{n,i}}{2} + \nabla \delta p^{n,i}, \bar{u}^{n,i} \cdot \nabla v) \\ +\delta_2(\nabla \cdot \frac{\delta u^{n,i}}{2}, \nabla \cdot v) = -r_u^{n,i}\end{aligned} \tag{2.2}$$

Die rechte Seite ist definiert als:

$$\begin{aligned}r_u^{n,i} :=& (u^{n,i} - u^{n-1}, v)k_n^{-1} \\ &+ (\bar{u}^{n,i} \cdot \nabla \bar{u}^{n,i}, v) \\ &+ \nu(\nabla \bar{u}^{n,i} : \nabla v) \\ &- (p^{n,i}, \nabla \cdot v) \\ &+ SD_\delta(\bar{u}^{n,i}, p^{n,i}; v, 0) \\ &- (f, v)\end{aligned}$$

Analog wird die Kontinuitätsgleichung behandelt:

$$\begin{aligned}(\nabla \cdot \frac{\delta u^{n,i}}{2}, q) + \delta_1(\frac{\delta u^{n,i}}{2} \cdot \nabla \bar{u}^{n,i}, \nabla q) + \delta_1(\bar{u}^{n,i} \cdot \nabla \frac{\delta u^{n,i}}{2}, \nabla q) \\ +\delta_1(\nabla \delta p^{n,i}, \nabla q) = -r_p^{n,i}\end{aligned} \tag{2.3}$$

Hier lautet die rechte Seite:

$$r_p^{n,i} := (\nabla \cdot \bar{u}^{n,i}, q) + SD_\delta(\bar{u}^{n,i}, p^{n,i}; 0, q)$$

Um eine kompakte Darstellung zu erhalten, wird analog zum vorherigen Kapitel die Matrixdarstellung von 2.2 und 2.3 formuliert. Schreibt man die Unbekannten $\delta u^{n,i}$ und $\delta p^{n,i}$ als Summe von Basisfunktionen und sucht die dazugehörigen Koeffizienten, so erhält man:

$$\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} \begin{pmatrix} \delta \mathbf{u}^{n,i} \\ \delta \mathbf{p}^{n,i} \end{pmatrix} = \mathbf{r}$$

Die rechte Seite $\mathbf{r}$ enthält die Koeffizienten von $r_u^{n,i}$ und $r_p^{n,i}$ bezüglich der Finite-Elemente-Basis. Da die Gleichungen 2.2 und 2.3 linear in den Unbekannten sind, hängen die Blöcke der Matrix nicht von der Lösung ab. In jeder Newton-Iteration ist also ein lineares Gleichungssystem zu lösen und anschließend die Newtonkorrektur zu berechnen.

Das Newton-Verfahren konvergiert lokal quadratisch falls die Jacobi-Matrix in der Lösung invertierbar ist. Allerdings ist der Konvergenzradius bei der Anwendung auf die Navier-Stokes-Gleichungen relativ klein und typischerweise proportional zu $\nu$ (siehe [12] oder [4, S. 328]). Das bedeutet, dass bei niedrigen Viskositäten gute Initialwerte benötigt werden, um Konvergenz zu garantieren. Eine mögliche Abhilfe besteht darin, zunächst einige Schritte mit einem stabileren Verfahren durchzuführen und dann die schnelle Konvergenz des Newton-Verfahrens zu nutzen. Dafür eignet sich zum Beispiel die Picard-Iteration, die nun vorgestellt wird.

## 2.2 Picard-Iteration

Die Picard-Iteration ist eine Fixpunktiteration bei der die nichtlinearen Terme, also der Konvektionsterm und die Stabilisierungsterme, an der zuletzt berechneten Iteration ausgewertet werden. Die Matrixdarstellung entspricht also der Gleichung 1.12, mit dem Unterschied, dass die Blöcke an der letzten Iteration ausgewertet werden:

$$\begin{pmatrix} F(u^{n,i}) & B^T(u^{n,i}) \\ B(u^{n,i}) & -C \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n,i+1} \\ \mathbf{p}^{n,i+1} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{2.4}$$

Die Blöck dieser Matrix sind wie in 1.13 definiert. Schreibt man 2.4 in der Variationsformulierung, so lautet die linearisierte Impulsgleichung

$$\begin{aligned}(u^{n,i+1} - u^{n-1}, v)k_n^{-1} + (\bar{u}^{n,i} \cdot \nabla \bar{u}^{n,i+1}, v) + \nu(\nabla \bar{u}^{n,i+1}, \nabla v) - (p^{n,i+1}, \nabla \cdot v) \\ +\delta_1(\bar{u}^{n,i} \cdot \nabla \bar{u}^{n,i+1} + \nabla p^{n,i+1} - f, \bar{u}^{n,i} \cdot \nabla v) + \delta_2(\nabla \cdot \bar{u}^{n,i+1}, \nabla \cdot v) = (f, v)\end{aligned}$$

und die Kontinuitätsgleichung:

$$(\nabla \cdot \bar{u}^{n,i+1}, q) + \delta_1(\bar{u}^n \cdot \nabla \bar{u}^{n+1} + \nabla p^{n,i+1}, \nabla q) = 0$$

Die Konvergenzgeschwindigkeit der Picard-Iteration ist im Gegensatz zum Newton-Verfahren lediglich linear, hat dafür aber einen deutlich größeren Konvergenzradius. Unter geeigneten Voraussetzungen lässt sich sogar globale Konvergenz beweisen, siehe z.B. [12] oder [24, S. 282]. Einen ausführlichen Vergleich zwischen Newton-Verfahren und Picard-Iteration bei der Anwendung auf die Navier-Stokes-Gleichungen findet man in [4, S. 370].

## 2.3 Splitting-Verfahren

Hoffmann et al. schlagen in [11] ein Splitting-Verfahren zur Lösung der mit der G2-Methode stabilisierten cG(1)cG(1)-Diskretisierung vor. Genauer handelt es sich, um ein Variante des Block-Gauß-Seidel-Verfahrens für nichtlineare Systeme.

Mit der Matrixdarstellung 1.12 als Ausgangsbasis, ist in jedem Zeitschritt ein Gleichungssystem der Form

$$\begin{pmatrix} F(u^n) & B^T(u^n) \\ B(u^n) & -C \end{pmatrix} \begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{2.5}$$

zu lösen. Dabei kommt wegen der Abhängigkeit der Blöcke $F$, $B^T$ und $B$ von $u^n$ die Nichtlinearität zustande. Ein Splitting-Verfahren teilt diesen Operator nun auf

$$\begin{pmatrix} F(u^n) & B^T(u^n) \\ B(u^n) & -C \end{pmatrix} = L + D + R$$

in die strikte untere Blockdreicksmatrix $L$, die Blockdiagonalmatrix $D$ und die strikte obere Blockdreicksmatrix $R$. Unter der Voraussetzung, dass $D + R$ invertierbar ist, erhält man durch einfaches Umstellen folgenden Fixpunktgleichung:

$$\begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix} = (D + R)^{-1}(\begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} - L \begin{pmatrix} \mathbf{u}^n \\ \mathbf{p}^n \end{pmatrix})$$

Bei Verwendung dieser Gleichung als Fixpunktiteration lautet das Iterationsverfahren:

$$\begin{pmatrix} F(u^{n,i}) & B^T(u^{n,i}) \\ 0 & -C \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n,i+1} \\ \mathbf{p}^{n,i+1} \end{pmatrix} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ B(u^{n,i}) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n,i} \\ \mathbf{p}^{n,i} \end{pmatrix}$$

Durch Ausmultiplizieren dieser Gleichung sieht man, dass eine Iteration in zwei Schritten durchgeführt werden kann:

1. $C\mathbf{p}^{n,i+1} = -\mathbf{r}_p + B(u^{n,i})\mathbf{u}^{n,i}$
2. $F(u^{n,i)})\mathbf{u}^{n,i+1} = \mathbf{r}_u - B^T(u^{n,i})\mathbf{p}^{n,i+1}$

Alternierend werden also zweite und erste Zeile des Ursprungsystems 2.5 nach $p^n$ und $u^n$ aufgelöst. Insgesamt besteht ein Iterationsschritt somit aus dem Lösen eines linearen Systems mit F und eines mit C sowie zwei Matrix-Vektormultiplikationen.

Bei dieser Vorgehensweise muss der Matrix $C$ besondere Aufmerksamkeit gewidmet werden, um deren Invertierbarkeit sicherzustellen. $C$ entspricht einer diskreten Poisson-Gleichung mit natürlichen Randbedingungen auf dem kompletten Rand (vgl. 1.13). Wie im kontinuierlichen Fall ist die Lösung eines diskreten Poisson-Problems mit natürlichen Randbedingungen nur bis auf eine Konstante bestimmt. Insbesondere kann $C$ nicht regulär sein. Es gibt verschiedene Ansätze die Lösbarkeit trotzdem sicherzustellen. Eine Möglichkeit ist die Störung von $C$ durch eine skalierte Identitätsmatrix. Das heißt man ersetzt $C$ durch $C+\epsilon I$ mit $\epsilon > 0$ und der Identitätsmatrix $I$, so dass die gestörte Matrix regulär ist. Eine andere Möglichkeit ist die Einführung einer künstlichen Dirichlet-Randbedingung auf dem Druckraum. Diese Methode wird später in den Benchmarks eingesetzt und dort genauer behandelt. Zu beachten ist jedoch, dass beide Ansätze das ursprüngliche System verändern und die Lösung damit verfälscht wird.

Für die Konvergenz des Splitting-Verfahrens spielt der Spektralradius von $(D + R)^{-1}L$ eine zentral Rolle. Aus dem Fixpunktsatz von Banach folgt nämlich, dass die Fixpunktiteration (linear) konvergiert, wenn der Spektralradius von $(D + R)^{-1}L$ unabhängig von $u^n$ kleiner als $1 - \epsilon$ mit $\epsilon > 0$ ist. Leider ist diese Bedingung bei Anwendung auf die Navier-Stokes-Gleichungen häufig nicht erfüllt, wie die Benchmarks später zeigen werden.

## 2.4 Abbruchkriterium

Das Abbruchkriterium entscheidet, wann die Iteration die Lösung ausreichend approximiert und das Iterationverfahren damit abgebrochen werden kann. Ein einfaches und effektives Abbruchkriterium ist die Norm des Abstandes zwischen zwei Iterationen. Das heißt, das Iterationverfahren läuft solange bis

$$(||u^{n,i+1} - u^{n,i}||^2 + ||p^{n,i+1} - p^{n,i}||^2)^{\frac{1}{2}} < \epsilon$$

mit einer geeigneten Toleranz $\epsilon > 0$ gilt. Da die Lösungen Funktionen aus $\mathcal{H}^1(\Omega)$ sind, bietet sich als Norm die Sobolewnorm an. Diese lautet im Fall $\mathcal{H}^1(\Omega)$:

$$||x||^2 = \sum_{|\alpha| \leq 1} ||\partial^\alpha x||^2_{L^2(\Omega)} = \sum_{|\alpha| \leq 1} \int_\Omega |\partial^\alpha x|^2$$

Ein alternatives Abbruchkriterium ist die Norm des Residuum. Keines der beiden Kriterien garantiert jedoch, dass die approximierte Lösung nahe an der tatsächlichen Lösung ist. Trotzdem werden sie in der Praxis erfolgreich eingesetzt.

# Kapitel 3

# Block-Vorkonditionierung

Bei Anwendung des Newton-Verfahrens oder der Picard-Iteration zur Lösung des nichtlinearen Systems muss in jedem Iterationsschritt ein lineares Gleichungssystem mit Blockstruktur gelöst werden:

$$A \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \mathbf{r} \qquad \text{mit} \quad A := \begin{pmatrix} F & B^T \\ B & -C \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} \mathbf{r}_u \\ \mathbf{r}_p \end{pmatrix} \tag{3.1}$$

Da die Basisfunktionen der Finite-Elemente-Räume einen kleinen Träger besitzen, handelt es sich bei den Blöcken der Matrix $A$ um dünnbesetzte Matrizen. Allerdings ist das Gleichungssystem in der Regel sehr groß und benötigt dementsprechend viel Sorgfalt bei der Wahl geeigneter Lösungsverfahren.

Ein erster Ansatz zur Lösung von 3.1 ist die Anwendung einer direkten Methode wie die LU-Zerlegung. Allerdings kann sie die dünnbesetzte Struktur der Matrix $A$ nur schlecht ausnutzen, wodurch der Rechen- und Speicheraufwand mit der Problemdimension stark ansteigt. Spezielle Varianten der LU-Zerlegung für dünnbesetzte Matrizen versuchen diese Problematik zu minimieren, trotzdem wird bei steigender Dimension schnell die Grenze der verfügbaren Rechnerkapazität erreicht.

Stattdessen werden iterative Methoden zur näherungsweisen Lösung von 3.1 verwendet. Zu den bekanntesten iterativen Verfahren zählen GMRES [25] und BiCGStab [32]. Beide gehören zur Klasse der Krylov-Unterraum-Verfahren und eignen sich sehr gut für große, nicht-symmetrische und dünnbesetzte Probleme. In einem Vergleich verschiedener Newton-Krylov Löser hat sich der GMRES als besonders geeignet herausgestellt [19]. Zudem hat GMRES gegenüber BiCGStab den Vorteil, dass das Residuum mit jeder Iteration garantiert kleiner wird. Um den Umfang der Untersuchungen in Grenzen zu halten, wird daher im weiteren nur GMRES betrachtet, obwohl sich prinzipiell beide Verfahren für die Lösung der auftretenden Probleme eignen.

Entscheidend für eine schnelle Konvergenz des GMRES-Verfahrens ist die Wahl eines guten Vorkonditionierers. Ein (rechter) Vorkonditionierer erweitert ein lineares Problem $Ax = f$ durch die Einführung einer invertierbaren Matrix $M$ zu $(AM^{-1})(Mx) = f$. Statt dem Ursprungsproblem wird zunächst das vorkonditionierte Problem $(AM^{-1})y = f$ mit GMRES gelöst. Anschließend erhält man die gesuchte

Lösung durch $x = M^{-1}y$. Ein guter Vorkonditionierer $M$ sollte daher einerseits die Kondition der Matrix $A$ verbessern und damit zu einer schnellen Konvergenz bei der Lösung des vorkonditionierten Systems führen (z.B. durch die Wahl $M \approx A$). Andererseits sollte $M^{-1}$ effizient berechenbar sein, da im zweiten Schritt die Matrix-Vektor Multiplikation $M^{-1}y$ notwendig ist. Man beachte, dass während der Berechnung nur $M^{-1}$ benötigt wird und nicht $M$.

Bei der Suche nach geeigneten GMRES-Vorkonditionierern für 3.1 hat sich herausgestellt, dass die Blockstruktur des linearen Operators $A$ gut genutzt werden kann. Der Grund ist, dass für die einzelnen Blöcke sehr effiziente Lösungsverfahren existieren, diese aber nicht auf die ganze Blockmatrix angewendet werden können. Fast alle im Folgenden vorgestellten Vorkonditionierer basieren daher auf der Block-LDU-Zerlegung der Matrix $A$:

$$\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} I & F^{-1}B^T \\ 0 & I \end{pmatrix} \tag{3.2}$$

Dabei ist $S$ das Schurkomplement und definiert als $S := BF^{-1}B^T + C$. Invertiert man die Faktoren der LDU-Zerlegung stellt man fest, dass das Inverse von $F$ und $S$ darin vorkommen. Während für das Lösen eines linearen Gleichungssystems mit $F$ effiziente Verfahren existieren, ist das Schurkomplement im Allgemeinen vollbesetzt und daher für große Probleme nicht lösbar. Viele Block-Vorkonditionierer verwenden daher Komponenten der LDU-Zerlegung 3.2 zusammen mit einer guten Approximation des Schurkomplements.

Bevor in den folgenden Abschnitten verschiedene Block-Vorkonditionierer vorgestellt werden, zeigt Abbildung 3.1 zur Übersicht das Lösungsverfahren mit dem Newton-Verfahren als nichtlinearen Löser und GMRES als linearen Löser (oder kurz Newton-GMRES).

## 3.1 Druck-Korrektur-Vorkonditionierung

Diese Vorkonditionierer stammen von den Druck-Korrektur-Verfahren, die ursprünglich als stationäre iterative Lösungsverfahren konzipiert worden sind. Um aus einem Druck-Korrektur-Verfahren einen Vorkonditionierer zu erhalten, interpretiert man es als Splitting-Verfahren. Bei einem Splitting-Verfahren wird die zu lösende Matrix zunächst in einen (invertierbaren) Splitting-Operator und eine Fehlermatrix aufgespalten. Das heißt, der Operator $A$ wird zerlegt in $A = M - R$. Umformung von 3.1 führt dann auf die Fixpunktiteration:

$$\begin{pmatrix} \mathbf{u}^{n+1} \\ \mathbf{p}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^{n} \\ \mathbf{p}^{n} \end{pmatrix} + M^{-1}(\mathbf{r} - A \begin{pmatrix} \mathbf{u}^{n} \\ \mathbf{p}^{n} \end{pmatrix}) \tag{3.3}$$

Für das Residuum $\mathbf{R}^n = \mathbf{r} - A(\mathbf{u}^n, \mathbf{p}^n)^T$ gilt dann:

$$\mathbf{R}^n = (I - AM^{-1})\mathbf{R}^{n-1} = (I - AM^{-1})^n \mathbf{R}^0$$

**Zeitschleife**

**for** n=1,...N **do:**

$u^{n,0} = u^{n-1} \quad p^{n,0} = p^{n-1}$

**nichtlineare Schleife**

**do:**

Erzeuge $\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix}$ und rechte Seite $\mathbf{r}$ wie in Kapitel 3.1

**lineare Schleife**

Löse $\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} \begin{pmatrix} \delta\mathbf{u}^{n,i} \\ \delta\mathbf{p}^{n,i} \end{pmatrix} = \mathbf{r}$ mit GMRES
und Vorkonditionierung bis das Residuum $\leq \epsilon_{\text{GMRES}}$ ist.

Berechne das Newtonupdate:
$\mathbf{u}^{n,i+1} = \mathbf{u}^{n,i} + \delta\mathbf{u}^{n,i}$
$\mathbf{p}^{n,i+1} = \mathbf{p}^{n,i} + \delta\mathbf{p}^{n,i}$

**while** $(||u^{n,i+1} - u^{n,i}||^2 + ||p^{n,i+1} - p^{n,i}||^2)^{\frac{1}{2}} \geq \epsilon_{\text{Newton}}$

Abbildung 3.1: Pseudocode des Newton-GMRES Algorithmus

Mit dem Banachschen Fixpunktsatz folgt, dass diese Fixpunktiteration linear konvergiert, falls für den Spektralradius $\rho(I - AM^{-1}) < 1$ gilt. Für eine schnelle Konvergenz der Fixpunktiteration muss $M^{-1}$ also eine gute Approximation von $A^{-1}$ sein. Tatsächlich lässt sich 3.3 auch als einfache Form eines Krylov-Unterraum-Verfahrens interpretieren, das auf $AM^{-1}$ angewendet wird [4, S. 177]. Verwendet man stattdessen ein besseres Krylov-Unterraum-Verfahren wie GMRES und nutzt $M$ als Vorkonditionierer, so ist eine Beschleunigung des Konvergenzverhaltens zu erwarten.

Im Folgenden werden drei dieser Vorkonditionierer von den ursprünglichen Druck-Korrektur-Verfahren abgeleitet. Die Interpretation der Druck-Korrektur-Verfahren als algebraische Splitting-Verfahren wurde in [23] und [15] behandelt.

### 3.1.1 SIMPLE-Vorkonditionierer

Wie schon erwähnt stammt der SIMPLE-Vorkonditionierer aus dem gleichnamigen Lösungsverfahren. Bei dem SIMPLE-Verfahren (Semi Implicit Method for Pressure Linked Equations) wird die Impuls- und Kontinuitätsgleichung entkoppelt gelöst. Durch approximatives Lösen der Impulsgleichung entsteht zunächst eine Zwischenlösung für die Geschwindigkeitsfunktion. Anschließend wird mithilfe der Kontinuitätsgleichung ein Update für Geschwindigkeits- und Druckfeld bestimmt, so dass die Massenerhaltung eingehalten wird. Die einzelnen Schritte des SIMPLE-Verfahrens lauten:

1. Wähle eine Initialwert $\mathbf{p}^n$

2. Löse $F\mathbf{u}^{n+\frac{1}{2}} = \mathbf{r}_u - B^T\mathbf{p}^n$
3. Löse $-(Bdiag(F)B^T + C)\delta\mathbf{p} = \mathbf{r}_p - B\mathbf{u}^{n+\frac{1}{2}} + C\mathbf{p}^n$
4. Berechne $\mathbf{u}^{n+1} = \mathbf{u}^{n+\frac{1}{2}} - diag(F)^{-1}B^T\delta\mathbf{p}$
5. Berechne $\mathbf{p}^{n+1} = \mathbf{p}^n + \alpha\delta\mathbf{p}$
6. Falls Konvergenzkriterium nicht erfüllt ist, wähle $\mathbf{p}^n = \mathbf{p}^{n+1}$ und gehe zu Schritt 2.

Der Parameter $\alpha \in (0, 1]$ in Schritt 5 dient als Dämpfungsfaktor für das Druckupdate und kann zur Konvergenzbeschleunigung verwendet werden. Wird das SIMPLE-Verfahren als Splitting-Verfahren interpretiert, das heißt bringt man es in die Form 3.3 (mit $\mathbf{u}^n = 0$ und $\alpha = 1$), so erhält man den SIMPLE-Vorkonditionierer

$$\begin{aligned} M_{\text{SIMPLE}} &= \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & 0 \\ 0 & -\tilde{S} \end{pmatrix} \begin{pmatrix} I & D^{-1}B^T \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} F & 0 \\ B & -\tilde{S} \end{pmatrix} \begin{pmatrix} I & D^{-1}B^T \\ 0 & I \end{pmatrix} \end{aligned}$$

mit $D = diag(F)$ der Diagonalen von $F$ und dem approximierten Schurkomplement $\tilde{S} = BD^{-1}B^T + C$. In dieser Form wird auch der Zusammenhang mit der LDU-Zerlegung 3.2 sichtbar. Der SIMPLE-Vorkonditionierer entsteht, indem die $LD$ Blöcke zusammengefasst werden und $F$ durch dessen Diagonale im $U$ Block und im Schurkomplement ersetzt wird. Das Inverse von $M_{\text{SIMPLE}}$ berechnet sich zu:

$$M^{-1}_{\text{SIMPLE}} = \begin{pmatrix} I & -D^{-1}B^T \\ 0 & \alpha I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\tilde{S}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix}$$

Aus der Differenz $A - M$ wird ersichtlich wie gut der Vorkonditionierer die Matrix $A$ approximiert. Mit $\alpha = 1$ gilt:

$$E_{\text{SIMPLE}} = A - M_{\text{SIMPLE}} = \begin{pmatrix} 0 & B^T - FD^{-1}B^T \\ 0 & 0 \end{pmatrix}$$

Falls die Diagonale $D$ die wesentlichen Eigenschaften von $F$ widerspiegelt, wird diese Fehlermatrix klein. Die folgende Variation des SIMPLE-Vorkonditionierers liefert in der Praxis bessere Ergebnisse [26, 34].

### 3.1.2 SIMPLEC-Vorkonditionierer

Eine Verbesserung des SIMPLE-Vorkonditionierers entsteht, wenn $D = diag(F)$ durch eine Matrix mit mehr Informationen aus $F$ ersetzt wird. Der SIMPLEC-Vorkonditionierer verwendet hierfür die Diagonalmatrix mit der Reihensumme der Absolutwerte von $F$ als Diagonalelemente, gekennzeichnet als $\sum |F|$.

Die restliche Vorgehensweise ist analog wie bei SIMPLE, das heißt der Vorkonditionierer lautet

$$M_{\text{SIMPLEC}} = \begin{pmatrix} F & 0 \\ B & -\tilde{S} \end{pmatrix} \begin{pmatrix} I & (\sum |F|)^{-1} B^T \\ 0 & I \end{pmatrix}$$

mit dem approximierten Schurkomplement $\tilde{S} = B(\sum |F|)^{-1} B^T + C$. Es gilt:

$$M^{-1}_{\text{SIMPLEC}} = \begin{pmatrix} I & -(\sum |F|)^{-1} B^T \\ 0 & \alpha I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\tilde{S}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix}$$

Wie bei SIMPLE ist $\alpha$ ein Dämpfungsparameter in $(0, 1]$. Die Fehlermatrix berechnet sich bei $\alpha = 1$ zu:

$$E_{\text{SIMPLEC}} = A - M_{\text{SIMPLEC}} = \begin{pmatrix} 0 & B^T - F(\sum |F|)^{-1} B^T \\ 0 & 0 \end{pmatrix}$$

### 3.1.3 SIMPLER-Vorkonditionierer

Das SIMPLER-Verfahren (Semi Implicit Method for Pressure Linked Equations Revised) [21, 14] ist eine weitere Variante von SIMPLE. Der Unterschied ist, dass zunächst eine Zwischenlösung für den Druck bestimmt und danach eine Zwischenlösung für die Geschwindigkeit berechnet wird. Schließlich wird mithilfe einer Projektion die Kontinuitätsgleichung erzwungen.

Mit $D = diag(F)$ der Diagonalen von $F$ und $\tilde{S} = BD^{-1}B^T + C$ die Approximation des Schurkomplements lautet das SIMPLER-Verfahren:

1. Löse $-\tilde{S}\mathbf{p}^{n+\frac{1}{2}} = \mathbf{r}_p - BD^{-1}((D - F)\mathbf{u}^n + \mathbf{r}_u)$
2. Löse $F\mathbf{u}^{n+\frac{1}{2}} = \mathbf{r}_u - B^T\mathbf{p}^{n+\frac{1}{2}}$
3. Löse $-\tilde{S}\delta\mathbf{p} = \mathbf{r}_p - Bu^{n+\frac{1}{2}}$
4. Berechne $\mathbf{u}^{n+1} = \mathbf{u}^{n+\frac{1}{2}} - D^{-1}B^T\delta\mathbf{p}$
5. Berechne $\mathbf{p}^{n+1} = \mathbf{p}^{n+\frac{1}{2}} + \delta\mathbf{p}$
6. Falls Konvergenzkriterium nicht erreicht wurde, setze $\mathbf{p}^{n+\frac{1}{2}} = \mathbf{p}^{n+1}$ und gehe zu Schritt 2.

Für die Matrixdarstellung des SIMPLER-Vorkonditionierer werden eine Reihe von Matrizen benötigt:

$$T = \begin{pmatrix} I & -D^{-1}B^T \\ 0 & \alpha I \end{pmatrix} \qquad M = \begin{pmatrix} F & 0 \\ B & -\tilde{S} \end{pmatrix}$$

$$M_L = \begin{pmatrix} F & B^T \\ 0 & -\tilde{S} \end{pmatrix} \qquad T_L = \begin{pmatrix} I & 0 \\ -BD^{-1} & I \end{pmatrix}$$

Die Variable $\alpha$ in $T$ ist wieder ein Dämpfungsparameter aus $(0, 1]$. Mit diesen Matrizen hat der SIMPLER als Vorkonditionierer die Gestalt [15]:

$$\begin{aligned} M_{\text{SIMPLER}}^{-1} =& TM^{-1} - TM^{-1}AM_L^{-1}T_L + M_L^{-1}T_L \\ =& (TM^{-1}(T_L^{-1}M_L - A) + I)M_L^{-1}T_L \end{aligned} \tag{3.4}$$

Die Form 3.4 ist allerdings aus numerischer Sichtweise ungünstig. Eine Verbesserung ergibt sich, wenn der Term $M^{-1}(T_L^{-1}M_L - A)$ explizit aufgestellt wird:

$$M^{-1}(T_L^{-1}M_L - A) = \begin{pmatrix} 0 & 0 \\ -\tilde{S}^{-1}B(D^{-1}F - I) & 0 \end{pmatrix} =: K$$

Dadurch reduziert sich der Aufwand unter anderem um das Lösen eines Gleichungssystems mit dem Operator $F$, das in der Invertierung der Matrix $M$ notwendig gewesen wäre. Die numerisch optimierte Version des SIMPLER-Vorkonditionierers lautet also

$$M_{\text{SIMPLER}}^{-1} = (TK + I)M_L^{-1}T_L$$

wobei die Matrizen $T$, $M_L$ und $T_L$ wie oben definiert sind. Obwohl die Fehlermatrix $A - M_{\text{SIMPLER}}$ in allen Blöcken Einträge ungleich Null enthält, werden wir später sehen, dass dieser Vorkonditionierer sehr gut Ergebnisse liefert.

## 3.2 Approximate-Commutator-Vorkonditionierung

Die Vorkonditionierer aus dem vorherigen Abschnitten stammen von klassischen Verfahren, die zunächst als eigenständige Löser entwickelt wurden. Die Approximate-Commutator-Vorkonditionierer sind dagegen speziell als Vorkonditionierer entworfen worden. Sie verwenden als Grundlage die DU Komponente der LDU-Zerlegung 3.2 und konzentrieren sich auf eine möglichst gute Approximation des Schurkomplements. Konkret lautet die Zerlegung

$$A = \begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}$$

wobei $S = BF^{-1}B^T + C$ das Schurkomplement ist. Als Vorkonditionierer $M$ wählt man die rechte Komponenten der Zerlegung mit einer geeigneten Approximation des Schurkomplements $\tilde{S}$, das heißt:

$$M = \begin{pmatrix} F & B^T \\ 0 & -\tilde{S} \end{pmatrix} \tag{3.5}$$

Das vorkonditionierte System hat dann die Gestalt:

$$AM^{-1} = \begin{pmatrix} I & 0 \\ BF^{-1} & S\tilde{S}^{-1} \end{pmatrix} \tag{3.6}$$

Mit $S = \tilde{S}$ sind alle Eigenwerte der Matrix 3.6 gleich 1 und weiter kann gezeigt werden, dass sie Jordanblöcke der maximalen Größe 2 besitzt [20]. Diese beiden Eigenschaften reichen aus, um zu zeigen, dass GMRES angewendet auf 3.6 innerhalb von zwei Iterationen konvergiert. Mit einer guten Approximation des Schurkomplements ist also eine schnelle Konvergenz des vorkonditionierten Systems zu erwarten.

Führt man eine Blockfaktorisierung von $M^{-1}$ durch, so sieht man, dass bei Anwendung des Vorkonditionierers ein lineares Problem mit $F$ und eines mit dem Schurkomplement gelöst werden muss:

$$M^{-1} = \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\tilde{S}^{-1} \end{pmatrix}$$

Während zur Lösung von $F^{-1}$ effiziente iterative Methoden wie Mehrgitterverfahren eingesetzt werden können, ist das Schurkomplement $S$ eine vollbesetzte Matrix und eine Invertierung bei großer Dimension nicht durchführbar. Das Ziel eines guten Approximate-Commutator-Vorkonditionierer ist also, mit $\tilde{S}$ einerseits eine gute Annäherung an das Schurkomplement zu definieren, die andererseits aber eine effiziente Anwendung des Vorkonditionierers ermöglicht.

### 3.2.1 Druck-Konvektions-Diffusions-Vorkonditionierer

Der Druck-Konvektions-Diffusions-Vorkonditionierer (PCD) [4] konstruiert eine Näherung des Schurkomplements mithilfe eines Kommutators. Die Herleitung basiert auf der Verwendung der Picard-Iteration als nichtlinearen Löser. Außerdem gehen wir zunächst von unstabilisierten Finite-Elementen aus. Das heißt, die Blockmatrix $A$ ist wie in Abschnitt 2.2 mit $\delta_1 = \delta_2 = 0$ definiert. Beide Einschränkungen können zum Schluss aber aufgehoben werden.

Unter den genannten Voraussetzungen besteht die Matrix $F$ aus einer diskreten Version des Konvektions-Diffusions Operators $\mathcal{L} = id\, k^{-1} - \nu\Delta + u \cdot \nabla$ auf dem Raum $X^h$. Dabei ist $u$ das Geschwindigkeitsfeld der letzten Picard-Iteration (vgl. 2.4). Es wird nun angenommen, dass ein analoger Operator $\mathcal{L}_p = (id\, k^{-1} - \nu\Delta + u \cdot \nabla)_p$ auf dem Druckraum $M^h$ existiert (obwohl nicht für alle Funktionen aus $M^h \subset L^2(\Omega)$ der Differentialoperator definiert ist). Betrachten wir nun den Kommutator von $\mathcal{L}$ und $\mathcal{L}_p$ mit dem Gradientenoperator: $\mathcal{E} := \mathcal{L}\nabla - \nabla\mathcal{L}_p$. Der Kommutator verschwindet, falls $u$ konstant ist. Das motiviert die Annahme, das $\mathcal{E}$ auch noch für glatte $u$ in einem gewissen Sinne klein ist. Damit erhält man durch Multiplikation mit dem Divergenzoperator:

$$\Delta\mathcal{L}_p^{-1} \approx \nabla \cdot \mathcal{L}^{-1}\nabla \tag{3.7}$$

Die diskrete Version von 3.7 hat die Gestalt:

$$(Q_p^{-1}A_p)(Q_p^{-1}F_p)^{-1} \approx (Q_p^{-1}B)(Q^{-1}F)^{-1}(Q^{-1}B^T) \tag{3.8}$$

$F$ bzw. $F_p$ sind das diskrete Analogon des Konvektions-Diffusions Operators auf dem Geschwindigkeits- bzw. Druckraum, $A_p$ der diskrete Laplace-Operator, $B^T$ entspricht

dem Gradientoperator und $B$ dem Divergenzoperator. Die Inversen der Massenmatrizen $Q$ auf dem Geschwindigkeitsraum und $Q_p$ auf dem Druckraum werden jeweils für die richtige Skalierung benötigt. Multiplikation von links mit $Q_p$ in 3.8 liefert schließlich:

$$A_p F_p^{-1} Q_p \approx B F^{-1} B^T$$

Auf der rechten Seite steht das Schurkomplement $S = BF^{-1}B^T$, da wegen $\delta_1 = 0$ auch $C = 0$ gilt. Damit erhält man eine Approximation des Schurkomplements für unstabilisierte Finite-Elemente:

$$\tilde{S}_{\mathrm{PCD}} := A_p F_p^{-1} Q_p \approx B F^{-1} B^T \tag{3.9}$$

Der PCD-Vorkonditionierer ist der Operator 3.5 mit der Schurkomplement-Approximation $\tilde{S}_{\mathrm{PCD}}$.

Was passiert aber bei Verwendung einer stabilisierten Diskretisierung, also falls $\delta_1 \neq 0 \neq \delta_2$ gilt? Die Matrizen $F$, $B$, $B^T$ enthalten dann zusätzliche Stabilisierungsterme und es gilt $C \neq 0$. Damit hat das Schurkomplement dann die Form $S = BF^{-1}B^T + C$. Trotzdem kann 3.9 auch für stabilisierte Diskretisierungen verwendet werden [4, S. 349], das heißt es gilt:

$$\tilde{S}_{\mathrm{PCD}} = A_p F_p^{-1} Q_p \approx B F^{-1} B^T + C$$

Da $F_p$ dem Operator $F$ im Druckraum entsprechen soll, müssen die Stabilisierungsterme in $F$ auch in den künstlichen Operator $F_p$ übertragen werden. Das heißt, $F_p$ ist ein stabilisierter Konvektions-Diffusions-Operator auf dem Druckraum.

Betrachten wir die Konstruktion der künstlich eingeführten Operatoren $A_p$, $F_p$ und $Q_p$ näher. Bei der Picard-Iteration hat $F$ im unstabilisierten Fall (d.h. $\delta_1 = \delta_2 = 0$) die Form:

$$F = (u^{n,i+1}, v)k_n^{-1} + (\bar{u}^{n,i} \cdot \nabla \frac{u^{n,i+1}}{2}, v) + \nu(\nabla \frac{u^{n,i+1}}{2} : \nabla v)$$

$F_p$ ist der gleiche Operator wie $F$, jedoch übertragen auf den Druckraum. Man erhält $F_p$ also durch Ersetzen von $u^{n,i+1}$ durch $p^{n,i+1}$ und Verwendung von Testfunktionen aus dem Druckraum:

$$F_p = (p^{n,i+1}, q)k_n^{-1} + (\bar{u}^{n,i} \cdot \nabla \frac{p^{n,i+1}}{2}, q) + \nu(\nabla \frac{p^{n,i+1}}{2} : \nabla q)$$

Die stabilisierte Version von $F_p$ enthält zusätzlich die $\delta_1$ und $\delta_2$ Terme:

$$\begin{aligned} F_p =& (p^{n,i+1}, q)k_n^{-1} + (\bar{u}^{n,i} \cdot \nabla \frac{p^{n,i+1}}{2}, q) + \nu(\nabla q : \nabla \frac{p^{n,i+1}}{2}) \\ &+ \delta_1(\bar{u}^{n,i} \cdot \nabla \frac{p^{n,i+1}}{2}, \bar{u}^{n,i} \cdot \nabla q) + \delta_2(\nabla \cdot \frac{p^{n,i+1}}{2}, \nabla \cdot v) \end{aligned}$$

Der diskrete Laplace-Operator $A_p$ ist definiert als

$$A_p = (\nabla p^{n,i+1}, \nabla q)$$

wobei die Stabilisierungsterme bei einer cG(1)cG(1)-Diskretisierung wegfallen. Die Massenmatrix $Q_p$ berechnet sich analog wie auf dem Geschwindigkeitsraum durch:

$$Q_p = (p^{n,i+1}, q)$$

Eine offene Frage ist die Wahl der Randbedingungen für die künstlichen Operatoren $F_p$ und $A_p$. Verschiedene Untersuchungen ergaben, dass es günstig ist, für Rand mit einströmendem Fluid eine Dirichlet-Randbedingung und auf dem Rest eine natürliche Neumann-Randbedingung festzulegen, siehe [10] und [4, S. 348]. Dabei spielt der eigentliche Wert der Dirichlet-Randbedingung keine Rolle, da er nur in die rechte Seite eingeht und diese bei der Erstellung des PCD-Vorkonditionierers nicht verwendet wird.

Wegen der für die Konstruktion notwendigen künstlichen Operatoren ist der PCD-Vorkonditionierer kein "Black-Box"-Algorithmus. Dadurch entsteht einerseits bei der Anwendungsentwicklung zusätzlicher Aufwand, aber auch die Assemblierung dieser Operatoren während der Laufzeit darf nicht unterschätzt werden. Da zudem die Frage der Randbedingungen nicht vollständig beantwortet ist, wird im nächsten Abschnitt eine Variante des PCD-Vorkonditionierers vorgestellt, der nur die Informationen des zu lösenden Gleichungssystems benötigt.

### 3.2.2 LSC-Vorkonditionierer

Der LSC-Vorkonditionierer erstellt die Matrix $F_p$ aus dem vorherigen Abschnitt durch Lösen eines Kleinste-Quadrate-Problems mit dem Kommutator $\mathcal{E}$. Damit kann der Vorkonditionierer vollautomatisch konstruiert werden, das heißt man erhält einen "Black-Box"-Algorithmus. Der LSC Algorithmus ist in seiner Grundform nicht für stabilisierte Diskretisierungen geeignet, weshalb nun $\delta_1 = \delta_2 = 0$ angenommen wird. Im nächsten Abschnitt wird dann eine Erweiterung vorgestellt die diese Einschränkung aufhebt.

Für die Herleitung des LSC-Vorkonditionierers wird das Schurkomplement ähnlich wie bei PCD extrahiert. Sei dafür $\mathcal{E}$ der Kommutator aus dem vorherigen Abschnitt: $\mathcal{E} := \mathcal{L}\nabla - \nabla\mathcal{L}_p$. Das diskrete Analogon lautet:

$$\mathcal{E}_h = (Q^{-1}F)(Q^{-1}B^T) - (Q^{-1}B^T)(Q_p^{-1}F_p) \tag{3.10}$$

Multiplikation von links mit $BF^{-1}Q$ und von rechts mit $F_p^{-1}Q_p$ zusammen mit der Annahme $\mathcal{E}_h \approx 0$ isoliert das Schurkomplement:

$$BF^{-1}B^T \approx BQ^{-1}B^TF_p^{-1}Q_p \tag{3.11}$$

Allerdings enthält 3.11 den künstlichen Operator $F_p$, der eine vollautomatische Konstruktion des Vorkonditionierers verhindert. Die Idee ist nun, $F_p$ so zu wählen, dass

der Kommutator $\mathcal{E}_h$ in 3.10 minimiert wird. Dieser Minimierungsprozess von $\mathcal{E}_h$ wird spaltenweise durchgeführt. Das heißt, jede Spalte $[F_p]_j$ wird so gewählt, dass der Term

$$||[Q^{-1}FQ^{-1}B^T]_j - Q^{-1}B^TQ_p^{-1}[F_p]_j||_Q \tag{3.12}$$

minimal ist. Dabei ist $||\cdot||_Q$ die induzierte Norm des diskretisierten $L_2$-Skalarprodukts auf dem Geschwindigkeitsraum: $(u,v)_Q = (Qu,v)$. Die Lösungen für 3.12 findet man mittels der Normalgleichungen. Sie lauten:

$$Q_p^{-1}BQ^{-1}B^TQ_p^{-1}[F_p]_j = [Q_p^{-1}BQ^{-1}FQ^{-1}B^T]_j$$

Somit kann $F_p$ geschrieben werden als:

$$F_p = Q_p(BQ^{-1}B^T)^{-1}(BQ^{-1}FQ^{-1}B^T)$$

Substitution von $F_p$ in 3.11 führt schließlich auf folgende Approximation des Schurkomplements:

$$BF^{-1}B^T \approx (BQ^{-1}B^T)(BQ^{-1}FQ^{-1}B^T)^{-1}(BQ^{-1}B^T)$$

Eine kleine Modifikation ist notwendig, um den LSC-Vorkonditionierer zu erhalten. Für viele Diskretisierungen ist das Inverse der Massenmatrix $Q^{-1}$ eine dichte Matrix und daher schwierig lösbar. Mit dem sogenannten "mass lumping" wird daher $Q$ durch eine einfache Matrix $\hat{Q}$ ersetzt. Eine gängige Wahl ist die Einschränkung auf die Diagonalelemente $\hat{Q} = diag(Q)$. Insgesamt approximiert LSC das Inverse des Schurkomplements durch:

$$\tilde{S}_{\text{LSC}}^{-1} = (B\hat{Q}^{-1}B^T)^{-1}(B\hat{Q}^{-1}F\hat{Q}^{-1}B^T)(B\hat{Q}^{-1}B^T)^{-1}$$

### 3.2.3 LSC-Vorkonditionierer für stabilisierte Diskretisierungen

Elman et al. veröffentlichte 2007 eine Erweiterung des LSC-Vorkonditionierers für stabilisierte Finite-Elemente-Diskretisierungen (SLSC) [3]. Die Methode sieht allerdings nur eine Stabilisierung durch die Einführung des $C$ Blocks in der zu lösenden Blockmatrix vor. Die hier eingesetzte G2-Methode stabilisiert dagegen zusätzlich die Matrizen $F$, $B$ und $B^T$ (siehe 1.13). Im Rahmen dieser Arbeit wurden geeignete Skalierungsparameter für den SLSC-Vorkonditionierer entwickelt, die den Einsatz von SLSC mit der G2-Stabilisierung möglich machen. Zunächst wird aber die Grundform des SLSC-Vorkonditionierers vorgestellt.

**Definition 1** (SLSC-Vorkonditionierer)**.** *Das inverse Schurkomplement für stabilisierte uniforme Gitter kann approximiert werden durch*

$$\tilde{S}_{SLSC}^{-1} = \tilde{S}_{\gamma}^{-1} + \tilde{S}_{\alpha}^{-1}$$

*mit*

$$\tilde{S}_{\gamma}^{-1} = (BQ^{-1}B^T + \gamma\tilde{C})^{-1}(BQ^{-1}FQ^{-1}B^T)(BQ^{-1}B^T + \gamma\tilde{C})^{-1}$$

*und:*

$$\tilde{S}_\alpha^{-1} = \alpha D^{-1},\ D := diag(Bdiag(F)^{-1}B^T + \tilde{C})$$

*Dabei ist $\tilde{C}$ eine Stabilisierungsmatrix und $\alpha, \gamma \in \mathbb{R}$ sind Stabilisierungsparameter.*

Mit $\tilde{C} := C = -\delta_1(\nabla\psi, \nabla\psi)$ erhielte man den originalen SLSC-Vorkonditionierer. Für den Einsatz mit einer G2-Stabilisierung ist die Skalierung durch $\delta_1$ allerdings ungeeignet. Die in dieser Arbeit konstruierten Variante verwendet daher die unskalierte Matrix in SLSC:

$$\tilde{C} := -(\nabla\psi, \nabla\psi) = \frac{1}{\delta_1}C$$

Die in [3] ursprünglich empfohlenen Stabilisierungsparameter $\alpha$ und $\gamma$ liefern bei der G2-Methode keine guten Ergebnisse und müssen ebenfalls angepasst werden. Der Grund ist auch hier die zusätzliche Stabilisierung in den Matrizen $F$, $B$ und $B^T$. Ziel bei der Suche geeigneter Parameter $\alpha$ und $\gamma$ waren kleine Iterationszahlen, möglichst unabhängig von Problemgröße und Viskosität. Ein manuell durchgeführter Optimierungsprozess für die mit der G2-Methode stabilisierten cG(1)cG(1)-Diskretisierung ergab folgendes Ergebnis für $\alpha$:

$$\alpha = \frac{1}{\rho(Bdiag(F)^{-1}B^T D^{-1})}$$

Dabei bezeichnet $\rho$ den Spektralradius. Für $\gamma$ muss zwischen einem konvektionsdominanten und nicht-konvektionsdominanten Problem unterschieden werden. Ist die Strömung konvektionsdominant, wählt man

$$\gamma = \frac{d \cdot h\ \cdot \rho(F)}{6}$$

wobei $h$ die Gittergröße und d die Anzahl der Freiheitsgrad des Gitters ist. Ansonsten wird $\gamma$ gesetzt zu:

$$\gamma = \frac{\rho(F)}{9\nu}$$

Man beachte, dass die Parameter $\alpha$ und $\gamma$ rein experimentell ermittelt wurden. Obwohl die Wahl der Parameter schon so gute Ergebnisse liefern, gibt es mit Sicherheit noch Raum für Verbesserungen.

Tabelle 3.1 und 3.2 auf Seite 30 vergleicht die originalen SLSC-Vorkonditionierer mit den angepassten Parametern für die G2-Stabilisierung[1]. Eine genauere Untersuchung des SLSC-Vorkonditionierers mit den angepassten Parametern folgt in Kapitel 5.

[1] Als Testbeispiel diente das 2D-Benchmark aus Kapitel 5 mit CFL=0,1.

| Unbekannte | Original Parameter | Angepasste Parameter |
|---|---|---|
| $0,03 \cdot 10^6$ | $152,5$ | 41 |
| $0,13 \cdot 10^6$ | $> 400$ | 40 |
| $0,5 \cdot 10^6$ | $> 400$ | 40 |

Tabelle 3.1: Durchschnittlich notwendige GMRES Iterationen für die Lösung eines Problems der Form 3.1 mit dem originalen SLSC-Vorkonditionierer und mit angepassten Parametern. Die Viskosität ist $\nu = 10^{-5}$, d.h. konvektionsdominanter Fall.

| Unbekannte | Original Parameter | Angepasste Parameter |
|---|---|---|
| $0,03 \cdot 10^6$ | $> 400$ | 34 |
| $0,13 \cdot 10^6$ | $> 400$ | 33 |
| $0,5 \cdot 10^6$ | $> 400$ | 38 |

Tabelle 3.2: Wie Tabelle 3.1, aber mit $\nu = 1,0$, womit man den nicht konvektionsdominanten Fall erhält.

# Kapitel 4

# Softwareumgebung

Im Rahmen der Diplomarbeit wurden die in Kapitel 3 vorgestellten Block-Vorkonditionierer implementiert und getestet. Dieses Kapitel gibt eine kurze Einführung in die Softwareumgebung und die neu entwickelten Softwareklassen.

Als Softwaregrundlage dient das Paket Trilinos 9.0 [29]. Trilinos ist eine Sammlung numerischer Algorithmen, die alles enthält, um partielle Differentialgleichungen numerisch zu lösen. Die hierfür wichtigsten Pakete in Trilinos sind:

**Epetra** Epetra stellt Basisklassen für die Konstruktion und Manipulation von Matrizen und Vektoren bereit. Epetra ist parallelisierbar, was schnelle Berechnungen auf verteilten System erlaubt. Alle folgenden Pakete benutzen Epetra Datentypen als Basistyp.

**Thyra** Diese Paket enthält abstrakte Interfaces zu Vektorräumen, Vektoren und linearen Operatoren und wird von vielen Lösern in Trilinos verwendet. Die im Rahmen der Diplomarbeit implementierten Vorkonditionierer verwenden sowohl Thyra als auch Epetra Datentypen.

**AztecOO** AztecOO umfasst eine Menge von parallelen iterativen Lösern und Vorkonditionierern. Es enthält insbesondere Krylov-Unterraum-Verfahren wie GMRES, GMRESR und BiCGStab.

**Amesos** Amesos besteht aus einer Sammlung von Direktlösern für dünnbesetzte lineare Probleme. Es unterstützt auch externe Bibliotheken wie SuperLU_dist [17], eine schnelle parallele LU-Zerlegung, die im Folgenden als Direktlöser verwendet wird.

**ML** ML ist ein Paket für algebraische Mehrgittervorkonditionierung. Auf die Mehrgitterverfahren wird in Kapitel 6.3 genauer eingegangen.

**Stratimikos** Das Stratimikos Paket stellt einen einheitlichen Zugriff zu linearen Lösungsverfahren und Vorkonditionierern zur Verfügung. Mit Stratimikos ist es einfach, eine große Menge von verschiedenen Lösungsverfahren auszuprobieren. Insbesondere unterstützt Stratimikos die Pakete AztecOO, Amesos und ML.

In dieser Arbeit wird Stratimikos benutzt, um die auftretenden Teilprobleme in den Vorkonditionierern zu lösen.

**Meros** Meros ist ein Paket für Block-Vorkonditionierer in Trilinos. Die im Rahmen der Diplomarbeit entwickelten Vorkonditionierer erweitern das Meros Paket.

**Sundance** Sundance ist ein System für die Lösung von partiellen Differentialgleichungen. Es enthält insbesondere einen Finite-Elemente Code, der in dieser Arbeit verwendet wird, um die linearen Probleme zu erzeugen.

Die Navier-Stokes-Gleichungen werden in Trilinos folgendermaßen gelöst. Zunächst wird die schwache Form der linearisierten Navier-Stokes-Gleichungen in Sundance beschrieben. Dann müssen ein Gitter, die gewünschten Anfangs- und Randbedingungen und der zu verwendende Finite-Elemente-Raum angegeben werden. Mit diesen Informationen kann der Assembler von Sundance das lineare Gleichungssystem aufstellen. Je nach gewählter Linearisierung der Navier-Stokes-Gleichungen entspricht das genau den linearen Problemen, die in Kapitel 2 erarbeitet wurden.

Bei Verwendung des Newton-Verfahrens oder der Picard-Iteration werden die dabei entstehenden Gleichungssysteme mit GMRESR[1] und Vorkonditionierung gelöst. Der Vorkonditionierer wird mithilfe des Meros Pakets erstellt. Dieser wird dann, zusammen mit dem linearen Problem selbst, an das Paket AztecOO weitergegeben. AztecOO löst schließlich das System iterativ mit GMRES unter Verwendung des Vorkonditionierers.

Es folgt nun eine Beschreibung der neu entwickelten Vorkonditionierer in dem Paket Meros. Obwohl der PCD- und SIMPLE-Vorkonditionierer schon vorhanden waren, wurden sie von Grund auf neu implementiert, um eine einheitliche Schnittstelle für den Anwender zur Verfügung zu stellen. Jeder Vorkonditionierer besteht aus einer *OperatorSource*-Klasse, die die für die Konstruktion notwendigen Operatoren als Epetra Matrizen speichert, und einer *PreconditionerFactory*-Klasse, mit dessen Hilfe der Vorkonditionierer konstruiert wird. Das Meros Paket wurde um folgenden Vorkonditionierer erweitert:

**Meros_SIMPLE** Enthält den SIMPLE- und SIMPLEC-Vorkonditionierer. Bei der Anwendung werden die Klassen *SIMPLEOperatorSource* und *SIMPLEPreconditionerFactory* benutzt. Der Konstruktor von *SIMPLEOperatorSource* erwartet die Matrizen $F$, $B$ und $B^T$ und $C$ als Eingabe. Dem *SIMPLEPreconditionerFactory*-Konstruktor muss für die Invertierung von $F$ und $\tilde{S}$ jeweils ein Lösungsverfahren angegeben werden. Zusätzlich hat der Benutzer die Möglichkeit zwischen SIMPLE und SIMPLEC zu wählen und den Dämpfungswert $\alpha$ zu setzen. Fehlen diese Angaben, wird standardmäßig SIMPLEC und $\alpha = 1,0$ ausgewählt.

**Meros_SIMPLE2** Meros_SIMPLE2 entspricht Meros_SIMPLE mit dem Unterschied, dass das Schurkomplement nicht explizit ausgerechnet wird. Das erspart

[1] Eine Variante von GMRES, auf die später näher eingegangen wird

Rechenzeit, hat allerdings den Nachteil das keine direkten Löser und kein Mehrgitterverfahren verwendet werden können (siehe Kapitel 7). Im Normalfall, sollte man daher die Meros_SIMPLE bevorzugen. Die *OperatorSource*-Klasse ist *SIMPLEOperatorSource*. Der Konstruktor *SIMPLE2PreconditionerFactory* verwendet die gleichen Parameter wie bei Meros_SIMPLE.

**Meros_SIMPLER** Der SIMPLER-Vorkonditionierer benötigt für die Konstruktion zusätzlich die Diagonalmatrix der Massenmatrix $Q$. Daher erwartet *SIMPLEROperatorSource* neben $F$, $B$ und $B^T$ und $C$ auch einen Vektor, der die Diagonaleinträge von $Q$ enthält. Die Parameter von *SIMPLERPreconditionerFactory* sind analog wie bei Meros_SIMPLE (allerdings ohne die SIMPLEC Option).

**Meros_PCD** Der PCD-Vorkonditionierer in Meros. Da zusätzlich künstliche Operatoren bei der Konstruktion des Vorkonditionierers benötigt werden, erwartet *PCDOperatorSource* neben $F$, $B$, $B^T$ und $C$ auch die Matrizen $F_p$, $A_p$ und den Vektor $Q_p$. Da $F_p$ und $A_p$ bei der Konstruktion invertiert werden, benötigt der Konstruktor von *PCDPreconditionerFactory* jeweils Lösungsverfahren für Probleme mit diesen Operatoren.

**Meros_SLSC** Für die Konstruktion des SLSC-Vorkonditionierers erwartet *SLSCOperatorSource* die Operatoren $F$, $B$ und $B^T$ und $C$ und den Vektor mit den Diagonalelementen der Massenmatrix $Q$. Für die Konstruktion von SLSC müssen Lösungsverfahren für die Invertierung von $F$ und $BQ^{-1}B^T + \gamma\tilde{C}$ übergeben werden. Optional können auch die Parameter für die Eigenwertprobleme bei der Berechnung von $\alpha$ und $\gamma$ festgelegt werden. Die Standardvorgaben sind jedoch optimiert und funktionieren in den meisten Fällen zufriedenstellend.

Die Angabe der Lösungsverfahren in den *PreconditionerFactory*-Klassen läuft über das Trilinos Paket Stratimikos. Letztlich handelt es sich um eine Liste mit Parametern, die angeben welcher Lösungsalgorithmus mit welchen Einstellungen verwendet werden soll. Eine Liste mit gültigen Parametern findet sich in der Dokumentation von Stratimikos[2].

Betrachten wir zum Schluss exemplarisch die Erstellung des SIMPLEC-Vorkonditionierers mit Meros. Die *OperatorSource*-Klasse speichert alle Informationen, die zur Erstellung des Vorkonditionierers notwendig ist.

```
1  class SIMPLEOperatorSource : virtual public LinearOpSourceBase<double>
2      {
3      public:
4        /** \brief Default constructor. */
5        SIMPLEOperatorSource(Epetra_CrsMatrix* S00,
6                             Epetra_CrsMatrix* S01,
7                             Epetra_CrsMatrix* S10,
8                             Epetra_CrsMatrix* S11);
```

[2] http://trilinos.sandia.gov/packages/docs/dev/packages/stratimikos/doc/html/classStratimikos_1_1DefaultLinearSolverBuilder.html
(besucht am 13.6.2009)

```
9   [...]
10  }
```

Die Klasse *SIMPLEOperatorSource* erwartet also die vier Blockmatrizen $F$, $B^T$, $B$ und $C$. Diese Objekt wird später der *SIMPLEPreconditionerFactory*-Klasse übergeben:

```
1    class SIMPLEPreconditionerFactory
2      : public Thyra::PreconditionerFactoryBase<double>
3      {
4      public:
5        /* Constructor for the SIMPLE preconditioner factory.
6         * Takes an Stratimikos parameter list for  the F and Ms,
7         * a damping factor and a switch for the SIMPLEC preconditioner.
8         */
9        SIMPLEPreconditionerFactory(
10         RCP<Teuchos::ParameterList>   F_SolveStrategy,
11         RCP<Teuchos::ParameterList>   Ms_SolveStrategy,
12         double alpha,
13         bool SIMPLEC
14         );
15
16       /* Create an (uninitialized) LinearOperator object to be initalized as
             the
17        * preconditioner later in this->initializePrecOp().
18        */
19       RCP<Thyra::PreconditionerBase<double> > createPrec() const;
20
21       /* Initialize the SIMPLEPreconditioner object */
22       void initializePrec([...])
23       [...]
24 }
```

Der Konstruktor nimmt wichtige Parameter für die Erstellung des Vorkonditionierers entgegen. Zum einen muss ihm mitgeteilt werden, welche Lösungsverfahren für die Teilprobleme verwendet werden sollen. Im Fall von SIMPLEC sind ein Lösungsverfahren für $F^{-1}$ und $\tilde{S}^{-1}$ anzugeben. Zudem kann mit *alpha* der Dämpfungsfaktor festgelegt werden (vgl. 3.1.2). Der letzte Parameter gibt schließlich an ob der SIMPLE oder SIMPLEC Vorkonditionierer erstellt werden soll. Die *initializePrec* nimmt schließlich das *SIMPLEOperatorSource*-Objekt entgegen und konstruiert den Vorkonditionierer.

Eine typische Vorgehensweise zur Erzeugung des Vorkonditionierers lautet also:

```
1        // F, Bt, B, C are of type Epetra::Epetra_CrsMatrix
2        RCP<const LinearOpSourceBase<double> > mySLSCopSrcRcp = rcp(new
             SIMPLEOperatorSource(F,Bt,B,C));
3        // invFParams, invHParams are of type (Teuchos::RCP<Teuchos::
             ParameterList> paramList
4        RCP<PreconditionerFactoryBase<double> > merosPrecFac  = rcp(new
             SIMPLEPreconditionerFactory(invFParams, invHParams, 1.0, true));
5        RCP<PreconditionerBase<double> > Prcp    = merosPrecFac->createPrec();
6        merosPrecFac->initializePrec(mySLSCopSrcRcp, &*Prcp);
```

# Kapitel 5

# Benchmarks der Lösungsverfahren

Ziel dieses Kapitels ist es, die vorgestellten Lösungsverfahren für die stabilisierte cG(1)cG(1)-Diskretisierung der Navier-Stokes-Gleichungen zu vergleichen. Dafür berechnet jedes Lösungsverfahren einen Zeitschritt eines Benchmarkproblems und es werden die Iterationen bis zur Erreichung des Abbruchkriteriums gezählt. Bei Verwendung des Newton-Verfahrens oder der Picard-Iteration werden neben der Anzahl der Iterationen des nichtlinearen Lösers zusätzlich die Iterationen des (iterativen) linearen Löser gemessen (vergleiche mit Abbildung 3.1 auf Seite 21). Das Lösen eventuell auftretender Teilprobleme (wie z.B. lineare Probleme mit den Operatoren $F$, $\tilde{S}$, ...) wird in diesem Kapitel nicht untersucht.

Dass keine Zeitmessung durchgeführt wird, hat folgenden Grund: In jedem Lösungsverfahren treten jeweils unterschiedliche Teilprobleme auf. Bei einer Zeitmessung hängt die Laufzeit jedoch stark von der Lösungsdauer dieser Teilprobleme ab. Das heißt, für jedes Lösungsverfahren müssten zunächst optimale Löser für die Teilprobleme gefunden werden. Der Aufwand hierfür wäre viel zu groß. Stattdessen wird mithilfe der Iterationszahlen (die unabhängig vom Lösungsverfahren der Teilprobleme sind) ein gutes Lösungsverfahren ausgewählt. In Kapitel 6 werden dann für dieses Lösungsverfahren gute Löser für die auftretenden Teilprobleme gesucht.

## 5.1 Benchmarkproblem

Um die Lösungsverfahren miteinander vergleichen zu können, wird eine gemeinsames Benchmarkproblem benötigt. In dem DFG-Forschungsprojekt "Flow Simulation with high-perfomance computers" [31] wurde solch ein Problem für 2D und 3D vorgestellt, das in einer leicht veränderten Form als Grundlage dienen soll. Aufgabe ist es, in einem Kanal den Fluss um einen Kreis in 2D bzw. um einen Zylinder in 3D zu berechnen. Die genauen Abmessungen sind in Abbildung 5.1 bzw. 5.2 auf der nächsten Seite beschrieben. Man beachte, dass der Kreis (bzw. Zylinder) nicht genau mittig plaziert ist. Dadurch erwartet man, dass bei niedrigen Viskositäten unsymmetrische Turbulenzen in der Lösung auftreten. In dem Benchmark wird ohne Volumenkräfte gerechnet, das heißt es gilt $f = 0$ in 1.1.

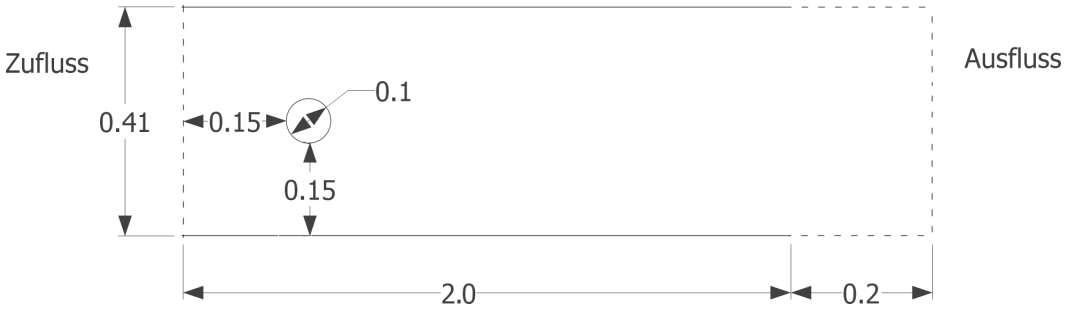


Abbildung 5.1: 2D-Benchmarkproblem

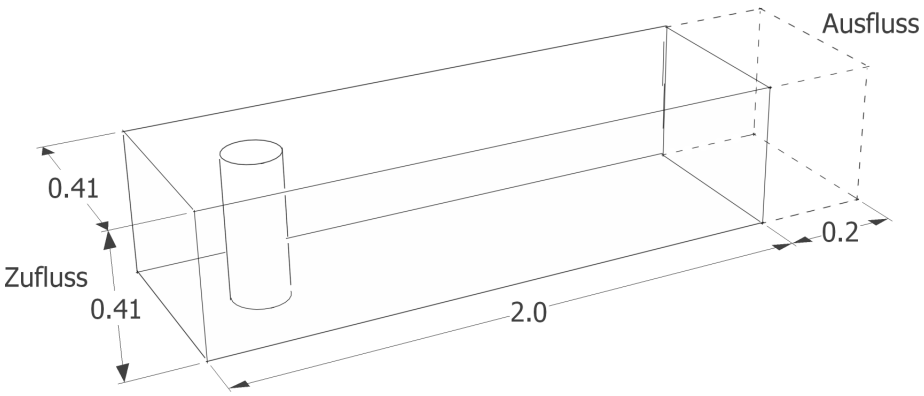


Abbildung 5.2: 3D-Benchmarkproblem

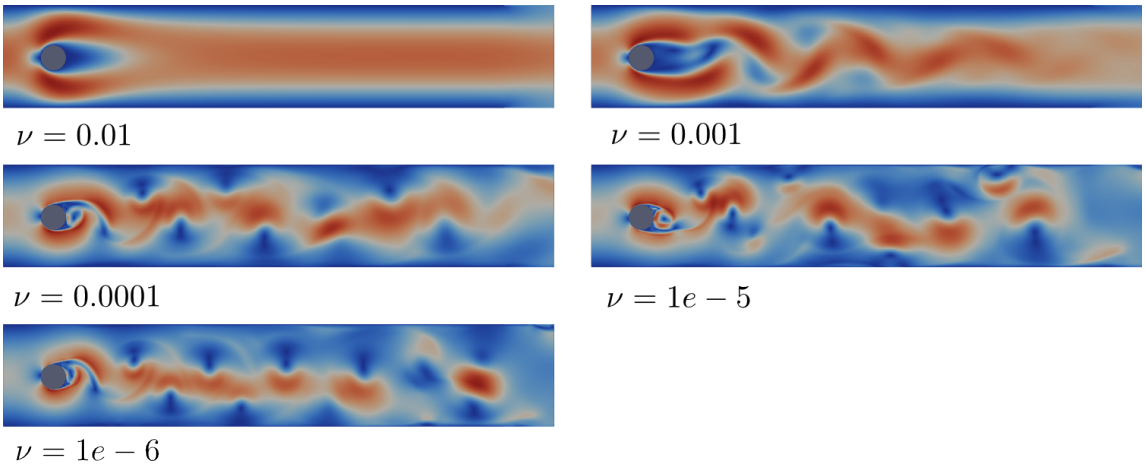


Abbildung 5.3: Lösung des 2D-Problems für verschiedene Viskositäten.

Von links strömt das Fluid mit einer definierten Geschwindigkeit in den Kanal hinein und auf der rechten Seite ohne Widerstand wieder hinaus. Die maximale Einflussgeschwindigkeit beträgt in 2D $1,5$ und in 3D $0,4$. An den Seitenwänden und dem Zylinder haftet das Fluid. Das heißt, es gelten Dirichlet-Randbedingungen am Einlass, den Seitenwänden und dem Zylinder. Das widerstandslose Herausfließen auf der rechten Seite wird durch natürliche Randbedingungen simuliert. Für die Diskretisierung wird das Gebiet mit einem Gitter versehen. Als Gitterelemente werden in 2D Dreiecke und in 3D Tetraeder verwendet. Die 2D-Gitter sind um den Kreis leicht verfeinert, in 3D sind sie uniform.

Für eine vollständige Beschreibung des Benchmarkproblems fehlt somit nur noch der Wert der Viskosität $\nu$. Da sich das Problem wesentlich mit der Viskosität ändert, werden die Benchmarks mit unterschiedlichen $\nu$-Werten durchgeführt. Die 2D-Lösungen der untersuchten Viskositäten zum Zeitpunkt $t = 2,0$ zeigt Abbildung 5.3. Es ist gut ersichtlich, wie die Turbulenzen mit sinkender Viskosität zunehmen. Um diese Turbulenzen numerisch auflösen zu können, werden sehr feine Gitter benötigt. Während die Probleme mit der dafür notwendigen Gitterweite in 2D noch berechenbar sind, sprengt der 3D-Fall die verfügbaren Rechenkapazitäten. Mit den hier verwendeten Gittern kann das 3D-Benchmarkproblem daher nur für $\nu \geq 10^{-4}$ berechnet werden. Für kleinere Viskositätswerte divergiert sowohl das Newton-Verfahren als auch die Picard-Iteration sobald Turbulenzen auftreten. Es sei noch angemerkt, dass eine Erhöhung der Stabilisierungsparameter $\kappa_1$ und $\kappa_2$ in der cG(1)cG(1)-Stabilisierung die Konvergenz wiederherstellen kann (vgl. Abschnitt 1.3). Die dadurch eingeführte künstliche Diffusion dämpft die auftretenden Turbulenzen allerdings so stark, dass sie in der Lösung nicht wiedererkennbar sind. Im restlichen Teil der Arbeit sind die Stabilisierungsparameter $\kappa_1 = \kappa_2 = 4.0$ gesetzt, womit eine gute Kombination von Stabilisierung und Lösungsgenauigkeit erreicht wird.

Für eine vollständige Analyse der betrachteten Lösungsverfahren werden sie unter unterschiedlichen Bedingungen getestet. Dabei werden folgenden Parameter variiert:

| Anzahl der Unbekannten | Knoten im 2D-Gitter | Knoten im 3D-Gitter |
|---|---|---|
| $33 \cdot 10^3$ | $11 \cdot 10^3$ | $8 \cdot 10^3$ |
| $131 \cdot 10^3$ | $44 \cdot 10^3$ | $33 \cdot 10^3$ |
| $524 \cdot 10^3$ | $175 \cdot 10^3$ | $131 \cdot 10^3$ |

Tabelle 5.1: Zusammenhang von Problemgröße und Anzahl der Gitterkonten.

- Die Feinheit des Gitters und resultierend daraus die Anzahl der Unbekannten im System. Für 2D und 3D werden jeweils drei verschiedene Gitter verwendet. Sie sind so konstruiert, dass die resultierende Anzahl der Unbekannten $33 \cdot 10^3$, $131 \cdot 10^3$ und $524 \cdot 10^3$ beträgt und seien im Folgenden durch 0,03M, 0,13M und 0,5M abgekürzt. Dabei berechnet sich die Anzahl der Unbekannten durch die Tatsache, dass bei einer cG(1)cG(1)-Diskretisierung pro Knoten 3 (bzw. 4) Unbekannte in 2D (bzw. 3D) gespeichert werden, nämlich die Geschwindigkeit in 2 (bzw. 3) Richtungen und der Druck. Der Zusammenhang zwischen Gitterknoten und Anzahl der Unbekannten ist in Tabelle 5.1 illustriert.

- Die Viskosität $\nu$. In 2D können mit den verwendeten Gittern Viskositäten bis zu $\nu = 10^{-6}$ untersucht werden. In 3D wird, wie schon erwähnt, nur bis $\nu = 10^{-4}$ gerechnet.

- Die CFL-Zahl. Sie ist definiert als:

$$\mathrm{CFL} = \max_{i=1..d} \frac{|u_i| \cdot \Delta t}{h}$$

  Bei gleicher Problemstellung beschreibt die CFL-Zahl im wesentlichen das Verhältnis zwischen Gittergröße $h$ und Zeitschrittweite $\Delta t$. Wie man aus den numerischen Ergebnissen ablesen kann, gilt für das 2D Benchmarkproblem die Näherung $\max |u| \approx 2$ und $\max |u| \approx 0,5$ in 3D. Damit kann aus einer gewählten CFL-Zahl die Zeitschrittweite berechnet werden. Zum Beispiel erhält man in 2D für $\mathrm{CFL} = 1,0$ und Gitter mit 0,5M den Wert $k_n = 1,2 \cdot 10^{-3}$.

  Bei gleichbleibendem Gitter ist die CFL-Zahl proportional zu dem Zeitschritt $k_n$. Ist die Aufgabe, die Lösung zu einem bestimmten Zeitpunkt zu berechnen, ist es günstig, große Zeitschritte zu wählen. Mit steigender Schrittweite sind die auftretenden Probleme jedoch schwieriger zu lösen (im Sinne von steigender Iterationszahlen der nichtlinearen und linearen Löser). Um Turbulenzen richtig simulieren zu können, ist bei der Wahl des Zeitschritts automatische eine obere Schranke gegeben.

Von einem guten Lösungsverfahren wird erwartet, dass die notwendigen Iterationen möglichst unabhängig von diesen Parametern sind. Folgende Lösungsverfahren werden unter diesem Gesichtspunkt untersucht:

- Splitting-Verfahren

- Newton-GMRESR-SIMPLEC,
  d.h. ein Newton-Verfahren für die nichtlineare Gleichung und GMRESR mit SIMPLEC-Vorkonditionierer für die linearen Gleichungen. GMRESR [33] ist eine Erweiterung von GMRES, die den Einsatz eines variablen Vorkonditionierers erlaubt, das heißt der Vorkonditionierer darf sich nach jedem Iterationsschritt ändern. Der Vorteil ist, dass damit der Vorkonditionierer $M$ nicht exakt gelöst werden muss. Insbesondere können die Teilprobleme bei der Konstruktion des Vorkonditionierers mit iterativen Verfahren gelöst werden.

- Newton-GMRESR-SIMPLER

- Newton-GMRESR-PCD

- Newton-GMRESR-SLSC

Mit dem Newton-Verfahren treten bei dem Benchmarkproblem keine Konvergenzprobleme auf. Da es im Gegensatz zur Picard-Iteration eine quadratische Konvergenz aufweist, wird auf die Untersuchung der Picard-Iteration als nichtlinearen Löser verzichtet.

Als Abbruchkriterium wird die in Abschnitt 2.4 vorgestellte Methode verwendet (Abstand zweier Iterationen). Die geforderte Toleranz für das nichtlineare System ist dabei $10^{-6}$. Die folgenden Ergebnisse sind zum Zeitschritt $t = 2,0$ gemessen worden.

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ | $\nu = 10^{-5}$ | $\nu = 10^{-6}$ |
|---|---|---|---|---|---|---|
| 0,03M | 0,1 | NC | NC | NC | NC | NC |
| | 1,0 | NC | NC | NC | NC | NC |
| | 5,0 | NC | NC | NC | NC | NC |
| 0,13M | 0,1 | NC | NC | NC | NC | NC |
| | 1,0 | NC | NC | NC | NC | NC |
| | 5,0 | NC | NC | NC | NC | NC |
| 0,5M | 0,1 | NC | NC | NC | NC | NC |
| | 1,0 | NC | NC | NC | NC | NC |
| | 5,0 | NC | NC | NC | NC | NC |

Tabelle 5.2: Splitting-Verfahren, 2D-Problem

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ |
|---|---|---|---|---|
| 0,03M | 0,01 | 27 | 201 | NC |
| | 0,1 | 17 | 81 | 112 |
| | 0,5 | NC | 119 | 89 |
| 0,13M | 0,01 | 6 | NC | NC |
| | 0,1 | 14 | 222 | 272 |
| | 0,5 | NC | NC | 170 |
| 0,5M | 0,01 | 6 | NC | NC |
| | 0,1 | 19 | NC | NC |
| | 0,5 | NC | NC | NC |

Tabelle 5.3: Splitting-Verfahren, 3D-Problem

## 5.2 Splitting-Vefahren

Bevor das Splitting-Verfahren eingesetzt werden kann, muss die in Abschnitt 2.3 angesprochene Problematik der Invertierbarkeit von $C$ geklärt werden. Hier wird der Ansatz mit der Einführung geeigneter Dirichlet-Randbedingungen verfolgt. Am rechten Auslass des Benchmarksproblems gelten die natürlichen Neumann-Randbedingungen, das heißt:

$$\nu \frac{\partial u}{\partial n} = p \cdot n$$

Da die betrachteten $\nu$-Werten klein sind, bietet es sich an, am Auslass $p = 0$ zu setzen. Es ist zu beachten, dass dies eine künstliche Forderung an den Druck ist und damit die Problemstellung im Allgemeinen verfälscht wird.

Die Tabellen auf Seite 40 zeigen die notwendigen Iterationen im Zeitschritt $t = 2,0$ für verschieden feine Gitter in 2D und 3D. Das Kürzel NC steht für "no convergence" und bedeutet, dass nach 500 Iterationen das Abbruchkriterium noch nicht erfüllt wurde. Wie aus es Tabellen sofort ersichtlich ist, konvergiert das Splitting-Verfahren in 2D in keinem der Testfälle. Auch in 3D wird nur unter günstigen Voraussetzungen eine Konvergenz erzielt.

Die wesentlichen Operationen bei der Anwendung des Splitting-Verfahrens sind neben Matrix-Vektor Multiplikationen die Lösung eines linearen Problem mit $F$ und eines mit $C$. Man beachte, dass die Matrix $C$ unabhängig von den Updates des Splitting-Verfahrens ist und somit während eines Zeitschritt konstant bleibt. Wird für $C$ nun am Beginn jedes Zeitschritts die LU-Zerlegung berechnet, so reduziert sich der Aufwand einer Iteration im wesentlichen auf die Invertierung von $F$. Leider ist die LU-Zerlegung bei großer Problemdimension sehr aufwändig. Stattdessen muss ein iteratives Verfahren eingesetzt werden, bei dem die Konstanz von $C$ keine Vorteile bringt.

Insgesamt scheidet das Splitting-Verfahren aufgrund der schlechten Konvergenzeigenschaften als Lösungsmethode für benchmarkähnliche Probleme in 2D und 3D aus.

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ | $\nu = 10^{-5}$ | $\nu = 10^{-6}$ |
|---|---|---|---|---|---|---|
| | 0,1 | 6 (4) | 6 (4) | 6 (4) | 6 (4) | 6 (4) |
| 0,03M | 1,0 | 11 (4) | 10 (4) | 10 (4) | 10 (4) | 10 (4) |
| | 5,0 | 17 (4) | 14 (5) | 15 (5) | 15 (5) | 15 (5) |
| | 0,1 | 7 (4) | 6 (4) | 7 (4) | 6 (4) | 6 (4) |
| 0,13M | 1,0 | 13 (4) | 11 (4) | 13 (4) | 10 (4) | 10 (4) |
| | 5,0 | 27 (4) | 17 (5) | 19 (6) | 16 (7) | 16 (7) |
| | 0,1 | 8 (4) | 7 (4) | 8 (4) | 7 (4) | 6 (4) |
| 0,5M | 1,0 | 16 (4) | 12 (4) | 14 (4) | 11 (4) | 11 (4) |
| | 5,0 | 37 (4) | 19 (5) | 23 (6) | Subsolver | Subsolver |

Tabelle 5.4: GMRESR-SIMPLEC, 2D-Problem

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ |
|---|---|---|---|---|
| | 0,01 | 6 (3) | 7 (3) | 7 (3) |
| 0,03M | 0,1 | 6 (3) | 8 (3) | 8 (3) |
| | 0,5 | 7 (4) | 9 (4) | 9 (4) |
| | 0,01 | 6 (3) | 7 (3) | 7 (3) |
| 0,13M | 0,1 | 6 (3) | 8 (3) | 8 (3) |
| | 0,5 | 7 (3) | 10 (3) | 9 (3) |
| | 0,01 | 6 (3) | 7 (3) | 7 (3) |
| 0,5M | 0,1 | 6 (3) | 8 (3) | 8 (3) |
| | 0,5 | 7 (3) | 10 (3) | 12 (4) |

Tabelle 5.5: GMRESR-SIMPLEC, 3D-Problem

## 5.3 Newton-GMRESR-SIMPLEC

Dieses und die folgenden Lösungsverfahren sind Newton-GMRESR Kombinationen mit verschiedenen Vorkonditionierern. Bei den zugehörigen Benchmark-Tabellen sind jeweils zwei Werte angegeben. Der Wert in Klammern gibt die notwendigen Newton-Iterationen im Zeitschritt $t = 2,0$ an. Da die in jeder Newton-Iteration entstehenden linearen Probleme mit GMRESR nur auf eine Genauigkeit von $10^{-5}$ gelöst werden, kann die Anzahl der Newton-Iterationen bei gleichen Parametern, aber unterschiedlichen Vorkonditionierern leicht variieren. Der davor stehende Wert gibt die durchschnittlich notwendigen Iterationen für das Lösen der linearisierten Gleichung mit dem vorkonditionierten GMRESR-Verfahren an. Der Wert ist über die Newton-Iterationen des Zeitschritts gemittelt. Dieses Ergebnis ist besonders wichtig, da es die Wirkungsweise des Vorkonditionierers offen legt. Werden mehr als 500 GMRESR-Iterationen für die Konvergenz eines Problems benötigt, bzw. divergiert GMRESR, so wird das mit "NC" gekennzeichnet. "Subsolver" bedeutet dagegen, dass eines der Teilprobleme während der Anwendung des Vorkonditionierers nicht gelöst werden konnte.

Seite 42 listet die Benchmarkergebnisse für den SIMPLEC-Vorkonditionierer auf. Der ebenfalls vorgestellte SIMPLE-Vorkonditionierer wird nicht getestet, da SIMPLEC bei Vortests grundsätzlich bessere Ergebnisse lieferte. Wie aus den Tabellen ersichtlich ist, werden mit SIMPLEC durchgehend niedrige GMRESR-Iterationen in 2D und 3D benötigt. Betrachten wir zunächst eine Änderung der Viskosität. Obwohl ab $\nu = 10^{-3}$ erste Turbulenzeffekte auftreten, bleiben die notwendigen Iterationszahlen in allen Fällen ungefähr konstant. Genauso gut schneidet SIMPLEC bei der Erhöhung der Problemdimension ab. Eine Versechzehnfachung der Unbekannten in 2D und 3D erhöht die notwendigen GMRESR-Iterationen praktisch nicht. Beachtenswert ist auch die Unabhängigkeit von der CFL-Zahl. Obwohl die Probleme bei hoher CFL-Zahl numerisch deutlich schwieriger sind, erhöhen sich Newton- und GMRESR-Iterationen nur leicht.

Bei Anwendung des SIMPLEC-Vorkonditionierers ist neben Matrix-Vektor Multiplikationen lediglich ein lineares Problem mit $F$ und eines mit $\tilde{S} = B(\sum |F|)^{-1}B^T + C$ zu lösen. Da die Lösung der linearen Probleme der zeitintensivste Teil bei der Anwendung der untersuchten Vorkonditionierer ist, gehört SIMPLEC zu den unaufwändigsten Vorkonditionierern.

Insgesamt liefert dieses Lösungsverfahren für eine große Anzahl an Parametern für 2D und 3D sehr gute Werte.

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ | $\nu = 10^{-5}$ | $\nu = 10^{-6}$ |
|---|---|---|---|---|---|---|
| | 0,1 | 3 (4) | 3 (4) | 3 (4) | 3 (4) | 3 (4) |
| 0,03M | 1,0 | 5 (4) | 5 (4) | 5 (4) | 5 (4) | 5 (4) |
| | 5,0 | 10 (4) | 10 (5) | 11 (5) | 11 (5) | 11 (5) |
| | 0,1 | 3 (4) | 3 (4) | 3 (4) | 3 (4) | 3 (4) |
| 0,13M | 1,0 | 5 (4) | 5 (4) | 6 (4) | 6 (4) | 6 (4) |
| | 5,0 | 10 (4) | 11 (5) | 14 (6) | 14 (7) | 14 (7) |
| | 0,1 | 3 (4) | 4 (4) | 4 (4) | 3 (4) | 3 (4) |
| 0,5M | 1,0 | 7 (4) | 5 (4) | 7 (4) | 7 (4) | 7 (4) |
| | 5,0 | 13 (4) | 12 (4) | 24 (6) | Subsolver | Subsolver |

Tabelle 5.6: GMRESR-SIMPLER, 2D-Problem

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ |
|---|---|---|---|---|
| | 0,01 | 4 (3) | 4 (3) | 4(3) |
| 0,03M | 0,1 | 4 (3) | 5 (3) | 5 (3) |
| | 0,5 | 4 (4) | 6 (4) | 6 (4) |
| | 0,01 | 4 (3) | 4 (3) | 4 (3) |
| 0,13M | 0,1 | 4 (3) | 4 (3) | 4 (3) |
| | 0,5 | 3 (3) | 5 (3) | 5 (3) |
| | 0,01 | 4 (3) | 4 (3) | 4 (3) |
| 0,5M | 0,1 | 3 (3) | 4 (3) | 4 (3) |
| | 0,5 | 3 (3) | 5 (3) | 5 (4) |

Tabelle 5.7: GMRESR-SIMPLER, 3D-Problem

## 5.4 Newton-GMRESR-SIMPLER

Die Benchmarkergebnisse für SIMPLER auf Seite 44 zeigen ähnliche gute Ergebnisse wie bei SIMPLEC. Auch hier ist die Anzahl der GMRESR-Iterationen unabhängig von der Problemdimension und Viskosität. Wie bei SIMPLEC stellt man eine leichte Abhängigkeit von der CFL-Zahl fest. Im direkten Vergleich von SIMPLEC und SIMPLER kommt letzterer mit weniger GMRESR-Iterationen aus. Die Newton-Iterationen sind dagegen, wie erwartet, gleich.

Im Vergleich zu SIMPLEC ist der SIMPLER aufwändiger in der Anwendung. Neben mehr Matrix-Vektor Multiplikationen müssen zwei lineare Gleichungssysteme mit $\tilde{S} = Bdiag(F)^{-1}B^T + C$ und eines mit $F$ gelöst werden.

Wie der SIMPLEC eignet sich der SIMPLER-Vorkonditionierer also sehr gut als Vorkonditionierer für die Benchmarkprobleme.

| | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ | $\nu = 10^{-5}$ | $\nu = 10^{-6}$ |
|---|---|---|---|---|---|---|
| | 0,1 | 84 (4) | 93 (4) | 91 (4) | 78 (4) | 78 (4) |
| 0,03M | 1,0 | 105 (4) | 137 (4) | 151 (4) | 112 (4) | 113(4) |
| | 5,0 | 126 (4) | 229 (5) | 241 (5) | 178 (5) | 175 (5) |
| | 0,1 | 129 (4) | 152 (4) | 168 (4) | 131 (4) | 131 (4) |
| 0,13M | 1,0 | 126 (4) | 219 (4) | NC | 179 (4) | 179 (4) |
| | 5,0 | 125 (4) | NC | NC | NC | NC |
| | 0,1 | 203 (4) | NC | NC | NC | NC |
| 0,5M | 1,0 | 148 (4) | NC | NC | NC | NC |
| | 5,0 | 128 (4) | NC | NC | NC | NC |

Tabelle 5.8: GMRESR-PCD, 2D-Problem

| | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ |
|---|---|---|---|---|
| | 0,01 | 88 (3) | 75 (3) | 57 (4) |
| 0,03M | 0,1 | 79 (3) | 73 (3) | 55 (3) |
| | 0,5 | 103 (4) | 122 (4) | 128 (4) |
| | 0,01 | 77 (3) | 103 (3) | 61 (3) |
| 0,13M | 0,1 | 95 (3) | 101 (3) | 79 (3) |
| | 0,5 | 128 (3) | 183 (3) | 186 (4) |
| | 0,01 | 71 (3) | 140 (4) | 109 (4) |
| 0,5M | 0,1 | 117 (3) | 130 (3) | 105 (5) |
| | 0,5 | 152 (3) | NC | NC |

Tabelle 5.9: GMRESR-PCD, 3D-Problem

## 5.5 Newton-GMRESR-PCD

Seite 46 stellt die Benchmarksergebnisse mit dem Newton-GMRESR-PCD Verfahren zusammen. Zunächst sind die durchgehend hohen Iterationszahlen auffällig. Im 2D-Fall erkennt man dabei zwei verschiedene Verhaltensmuster. Für $\nu = 10^{-2}$ sind die Iterationszahlen nur schwach abhängig von Problemdimension und CFL-Zahl. Wird die Strömung jedoch turbulent, also falls $\nu \leq 10^{-3}$, ist eine starke Abhängigkeit von der Problemdimension erkennbar.

Etwas niedrigere, aber immer noch hohe Werte liefert PCD in 3D. Tatsächlich wird auch bei einer Problemdimension von 0,5M fast immer Konvergenz erreicht, aber auch hier ist die Abhängigkeit von der Anzahl der Unbekannten sichtbar.

Es stellt sich die Frage, woher dieses schlechte Verhalten rührt. Betrachtet man nochmals die Herleitung von PCD in Abschnitt 3.2.1, so wird ersichtlich, dass diese auf der Picard-Iteration und nicht dem hier verwendeten Newton-Verfahren basiert. Bei zusätzlichen Test wurden die Benchmarks deshalb mit der Picard-GMRESR-PCD Methode durchgeführt. Leider wurden auch damit ähnliche hohe Iterationszahlen erzielt. Stattdessen bringt die Wahl einer anderen Diskretisierung eine wesentliche Verbesserung mit sich. Insbesondere mit stabilen Finiten-Elementen ohne Stabilisierung wurden deutlich weniger Iterationen benötigt. Das stimmt auch mit den Ergebnissen aus vorhergehenden Untersuchungen überein, die für nichtturbulente Strömungen und stabiler Diskretisierung niedrige und gitterunabhängige Iterationszahlen feststellten, siehe zum Beispiel [18, 5, 27].

Bei der Anwendung des PCD-Vorkonditionierers sind neben Matrix-Vektor Multiplikationen ein lineares System mit $A_p$ und eines mit $F$ zu lösen. Aufgrund der Diagonalgestalt ist $Q_p$ leicht invertierbar. Sieht man von der Konstruktion der künstlichen Matrizen $A_p$ und $F_p$ ab, ist eine Anwendung des PCD-Vorkonditionierers insofern relativ günstig. Dennoch zeigen die hohen Iterationszahlen und deren Abhängigkeit von den Problemparametern, dass der PCD-Vorkonditionierer nicht für das untersuchte Problem geeignet ist.

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ | $\nu = 10^{-5}$ | $\nu = 10^{-6}$ |
|---|---|---|---|---|---|---|
| | 0,1 | 32 (4) | 36 (4) | 41 (4) | 41 (4) | 41 (4) |
| 0,03M | 1,0 | 31 (4) | 65 (4) | 129 (4) | 138 (4) | 134 (4) |
| | 5,0 | 64 (4) | NC | NC | NC | NC |
| | 0,1 | 27 (4) | 30 (4) | 40 (4) | 40 (4) | 40 (4) |
| 0,13M | 1,0 | 34 (4) | 34 (4) | 281 (4) | 191 (4) | 201 (4) |
| | 5,0 | 43 (4) | NC | NC | NC | NC |
| | 0,1 | 30 (4) | 28 (4) | 34 (4) | 40 (4) | 40 (4) |
| 0,5M | 1,0 | 26 (4) | 40 (4) | 383 (4) | NC | NC |
| | 5,0 | 31 (4) | NC | NC | NC | NC |

Tabelle 5.10: GMRESR-SLSC, 2D-Problem

| Dim | CFL | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ |
|---|---|---|---|---|
| | 0,01 | 70 (3) | 105 (4) | 121 (4) |
| 0,03M | 0,1 | 16 (3) | 21 (3) | 19 (3) |
| | 0,5 | 15 (4) | 223 (4) | 23 (4) |
| | 0,01 | 43 (3) | 110 (4) | 115 (4) |
| 0,13M | 0,1 | 12 (3) | 18 (3) | 15 (3) |
| | 0,5 | 13 (3) | 20 (3) | 20 (3) |
| | 0,01 | 48 (4) | 121 (4) | 126 (4) |
| 0,5M | 0,1 | 10 (3) | 15 (3) | 15 (3) |
| | 0,5 | 11 (3) | 17 (3) | 18 (4) |

Tabelle 5.11: GMRESR-SLSC, 3D-Problem

## 5.6 Newton-GMRESR-SLSC

Bei der Herleitung des SLSC-Vorkonditionierers (Abschnitt 3.2.3) wurden spezielle Stabilisierungsparameter für die G2-Methode vorgestellt, die hier nun eingesetzt werden. Dabei muss bei der Wahl des Parameters $\gamma$ zwischen konvektionsdominantem und nicht-konvektionsdominantem Fluss unterschieden werden. Weil die in den Benchmarks betrachteten $\nu$-Werte relativ klein sind, wird nun der Einfachheit halber der konvektionsdominante Fall für $\gamma$ angenommen.

Die Tabellen auf Seite 48 präsentieren die Iterationszahlen mit SLSC. In 2D und niedrigen CFL-Zahlen zeigt der Vorkonditionierer von der Viskosität und der Dimension unabhängige Iterationszahlen. Mit Erhöhung der CFL-Zahl steigen die notwendigen Iterationen jedoch schnell an. In 3D überzeugt der SLSC-Vorkonditionierer fast durchgehend mit niedrigen Iterationszahlen. Auch hier ist die Unabhängigkeit von der Viskosität und der Anzahl der Unbekannten ersichtlich.

Wie beim PCD-Vorkonditionierer wird als Approximation der Massenmatrix deren Diagonale verwendet, was die Invertierung einfach macht. Damit bleiben bei Anwendung des SLSC Operators ein lineares Problem mit $F$ und zwei Probleme mit $BQ_u^{-1}B^T + \gamma C$ zu lösen.

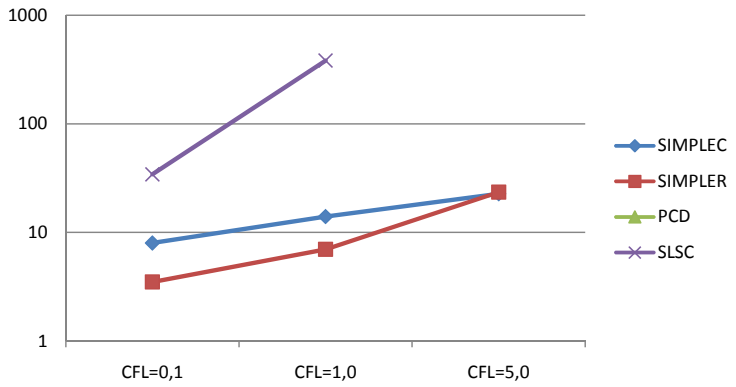


Abbildung 5.4: GMRESR-Iterationen für verschiedene CFL-Zahlen (2D-Problem, Dimension 0,5M und $\nu = 10^{-4}$).

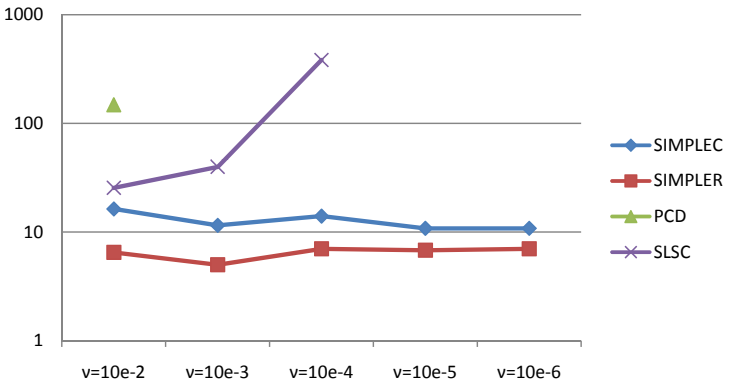


Abbildung 5.5: GMRESR-Iterationen für verschiedene $\nu$-Werte (2D-Problem, Dimension 0,5M und CFL=1,0).

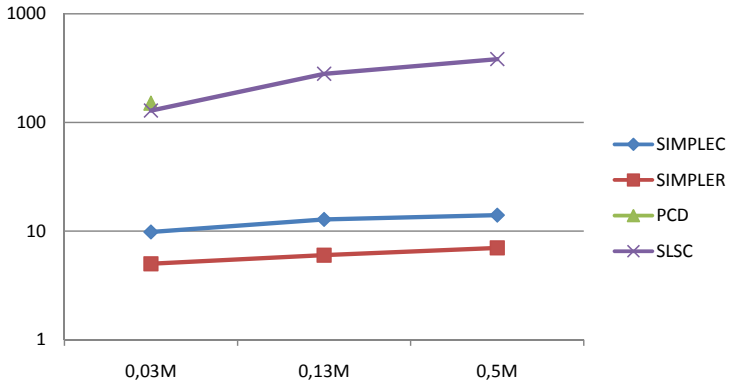


Abbildung 5.6: GMRESR-Iterationen für verschiedene Problemdimensionen (2D-Problem, CFL=1,0 und $\nu = 10^{-4}$).

## 5.7 Zusammenfassung

Es wurden fünf verschiedene Lösungsverfahren getestet. Dabei hatte sich das Splitting-Verfahren als ungeeignet herausgestellt. Die restlichen Verfahren sind vom Typ Newton-GMRESR und unterscheiden sich nur vom verwendeten Vorkonditionierer. Auf Seite 50 werden diese Vorkonditionierer direkt miteinander verglichen. Wie schon bemerkt, sind SIMPLEC und SIMPLER die einzigen Vorkonditionierer, die niedrige Iterationszahlen und Unabhängigkeit von den untersuchten Parametern aufweisen. Die etwas niedrigeren GMRESR-Iterationen des SIMPLER stehen der geringeren Komplexität des SIMPLEC-Vorkonditionierer gegenüber.

Damit stellt sich die Frage, welcher der beiden zu bevorzugen ist. Mit den optimierten Lösungsmethoden aus dem nächsten Kapitel erhält man in einem Testlauf für einen kompletten Zeitschritt folgende Lösungszeiten (mit dem 3D Benchmarkproblem, 0,5M Unbekannten, $\nu = 10^{-4}$ und CFL=0,1):

| | |
|---|---|
| Newton-GMRESR-SIMLEC: | 665s |
| Newton-GMRESR-SIMPLER: | 650s |

Hier muss der Anwender also selber entscheiden, welcher Vorkonditionierer er bevorzugt. Durch den einheitliche Implementierung in Meros ist ein Testen beider Verfahren schnell und einfach möglich. Man beachte dabei, dass der SIMPLEC-Vorkonditionierer ohne die Massenmatrix $Q$ auskommt, die bei der Konstruktion von SIMPLER benötigt wird.

# Kapitel 6

# Lösungsverfahren für die Teilprobleme

Dieses Kapitel untersucht Lösungsverfahren für die Teilprobleme, die bei der Anwendung der Vorkonditionierer entstehen. Beispielsweise muss bei Anwendung des SIMPLEC-Vorkonditionierers ein lineares System mit $F$ und eines mit der Approximation des Schurkomplements gelöst werden. Ziel ist es, eine optimale Lösungsstrategie für das Benchmarkproblem aus Kapitel 5 zu entwickeln. Dabei beschränken wir uns auf die im Kapitel 5 erfolgreichen Vorkonditionierer, nämlich SIMPLEC und SIMPLER.

Während bei den bisherigen Benchmarks die Rechenzeit außer Acht gelassen wurde, spielt sie nun eine signifikante Rolle. Dafür wird der Begriff Skalierbarkeit eingeführt: Ein Algorithmus heißt gut skalierbar, wenn der Rechenaufwand proportional mit der Problemgröße ansteigt. Für ein effizientes Lösungsverfahren ist aber neben der Skalierbarkeit auch die Parallelisierbarkeit entscheidend. Dabei wird versucht, das Problem in viele kleine, möglichst unabhängige Teilprobleme aufzuteilen, die dann parallel berechnet werden können. Nur mit Hilfe einer guten Parallelisierung und dem Einsatz von vielen vernetzten Rechnern ist es möglich, heute auftretende Probleme zu lösen. Um gleichzeitig die Skalierbarkeit und Parallelisierbarkeit eines Algorithmus zu testen, wird bei einer n-fachen Vergrößerung der Problemdimension die n-fache Anzahl an Prozessoren verwendet, das heißt:

$$\frac{\text{Problemdimension}}{\text{Anzahl der CPUs}} = const$$

Im Idealfall eines perfekt skalierbaren und parallelisierbaren Algorithmus ist die Ausführungsgeschwindigkeit dann unabhängig von der Problemgröße. Sobald jedoch Kommunikation zwischen den Prozessoren erforderlich wird, hat die Erhöhung der Problemgröße auch eine Laufzeiterhöhung zur Folge. Die Berechnungen in diesem Kapitel wurden auf einem Rechencluster durchgeführt. Dabei besteht jeder Rechner aus 8 x 2,7GHZ Quad-CPUs (AMD Opteron 8384) und 660GB Arbeitsspeicher. Insgesamt hat also jeder Rechner 32 CPUs. Diese Tatsache ist deshalb wichtig, da für Benchmarks mit mehr als 32 CPUs die Latenz der langsamen Netzwerkkommunika-

tion berücksichtigt werden muss.

Betrachten wir die zu lösenden Teilproblem bei der Verwendung von SIMPLEC bzw. SIMPLER als Vorkonditionierer. In jeder GMRESR-Iteration wird ein Matrix-Vektor Produkt mit $M^{-1}_{\text{SIMPLEC}}$ bzw. $M^{-1}_{\text{SIMPLER}}$ durchgeführt. Dabei müssen lineare Problem mit $F$ und $\tilde{S}$ gelöst werden (vgl. Abschnitt 3.1.2 und 3.1.3). Jedoch unterscheidet sich der Aufbau der Matrix $\tilde{S}$, es gilt nämlich $S_{\text{SIMPLEC}} := B(\sum |F|)^{-1}B^T + C$ und $S_{\text{SIMPLER}} := Bdiag(F)^{-1}B^T + C$. Dennoch ist die Struktur beider Matrizen identisch, da $\sum |F|$ und $diag(F)$ Diagonalmatrizen sind. Man erwartet daher, dass sich ein gutes Lösungsverfahren für beide Probleme gleichermaßen eignet. Aus diesem Grund wird im weiteren nur $F$ und $S_{\text{SIMPLEC}}$ untersucht, die Ergebnisse können aber auch direkt für den SIMPLER-Vorkonditionierer verwendet werden.

Es werden zwei Lösungsmethoden für die Teilprobleme untersucht: die LU-Zerlegung und das algebraische Mehrgitterverfahren. Um die Analyse übersichtlich zu halten, beschränken wir uns auf den 3D-Benchmark aus Kapitel 5. Die erarbeiteten Ergebnisse lassen sich jedoch auf den 2D-Fall übertragen.

## 6.1 LU-Zerlegung

Die LU-Zerlegung oder Gauß-Elimination ist das einfachste und bei kleiner Problemdimension sehr effektive Verfahren zur Lösung eines linearen Gleichungssystem der Form

$$A\mathbf{x} = \mathbf{b}$$

mit $A \in \mathbb{R}^{n\times n}$ invertierbar, $\mathbf{x} \in \mathbb{R}^n$ und $\mathbf{b} \in \mathbb{R}^n$. Die Grundidee ist, die Matrix durch elementare Umformungen auf Dreiecksgestalt zu bringen. Im Allgemeinen wird zusätzlich eine Pivotisierung und Skalierung benötigt, um einen numerisch stabilen Algorithmus zu erhalten. Die LU-Zerlegung hat dann die Form:

$$P_r D_r A D_c P_c = LU \tag{6.1}$$

Hier ist $L$ eine untere linke Dreiecksmatrix mit Diagonalelementen 1 und $U$ eine obere rechte Dreiecksmatrix. $P_r$ und $P_c$ sind Permutationsmatrizen für die Pivotisierung. Sie spielen außerdem bei dünnbesetzten Matrizen eine wichtige Rolle, da dann mit einer günstigen Wahl von $P_r$ und $P_c$ versucht wird, die Dünnbesetztheit auf die LU-Zerlegung zu übertragen. Die Diagonalmatrizen $D_r$ und $D_c$ skalieren die Zeilen und Spalten von A mit dem Ziel die numerische Stabilität zu verbessern. Typischerweise werden sie so gewählt, dass der betragsmäßig größte Eintrag jeder Zeile und Spalte von $D_r A D_c$ den Wert 1 hat.

Mit der Zerlegung 6.1 kann die Lösung effizient ausgerechnet werden:

$$\mathbf{x} = A^{-1}\mathbf{b} = (D_r^{-1}P_r^{-1}LUP_c^{-1}D_c^{-1})^{-1}\mathbf{b} = D_c(P_c(U^{-1}(L^{-1}(P_r(D_r\mathbf{b})))))$$

Eine Multiplikation von $P_r$ und $P_c$ entspricht einem Vertauschen der Zeilen bzw. Spalten. Durch die Dreiecksgestalt von $U$ und $L$ können die Terme $L^{-1}(P_r(D_r\mathbf{b})$ und $U^{-1}(L^{-1}(P_r(D_r\mathbf{b}))$ durch Substitution gelöst werden.

Die Pivotisierung mit $P_r$ und $P_c$ ist notwendig, um Nullelemente in der Diagonale zu vermeiden, da sonst eine Division durch 0 auftreten würde. Aber auch sehr kleine Pivotelemente führen zu betragsmäßig großen Werten in der Zerlegung und damit zu numerischen Instabilitäten. Eine gängige Wahl des Pivotelements ist daher, den betragsmäßig größten Eintrag in der noch zu zerlegenden Restmatrix zu wählen (Totalpivotisierung). Mit dieser Vorgehensweise können $P_r$ und $P_c$ aber erst im Laufe der Zerlegung erstellt werden. In einer parallelen Umgebung erfordert dies eine aufwändige Synchronisierung der Datenstrukturen und Neuverteilung der Aufgaben. Stattdessen wird in diesem Fall eine statische Pivotisierung [16] verwendet, bei der $P_r$ und $P_c$ schon während der Initialisierungsphase bestimmt werden. Häufig verfolgt man das Ziel, dass nach der Permutierung die betragsmäßig größten Elemente auf der Diagonale stehen.

Mit einer statischen Pivotisierung kann die LU-Zerlegung nun effizient parallel ausgeführt werden. Dafür wird die Matrix $\tilde{A} := P_r D_r A D_c P_c$ in $N \times N$ Blöcke zerlegt und an die verfügbaren Prozessoren verteilt. Algorithmus 1 zeigt die LU-Zerlegung in Matlab-Notation.

**Algorithmus 1** : Parallele LU-Zerlegung

**Input** : $N \times N$ Blockmatrix $\tilde{A}$
**Output** : Matrix L, U mit $\tilde{A} = LU$
**for** $K = 1 : N$ **do**
  Berechne die Faktorisierungsblöcke $L(K:N;K)$.
  Berechne parallel:
    $U(K;K+1:N) = L(K;K)^{-1}\tilde{A}(K;K+1:N)$.
  **for** $J = K+1 : N$ **do**
    **for** $I = K+1 : N$ **do**
      Berechne parallel: $\tilde{A}(I;J) = \tilde{A}(I;J) - L(I;K) \cdot U(K;J)$

Die Schritte in der äußersten for-Schleife sind nur bedingt parallelisierbar. Eine Verbesserung bietet das Piplining, bei dem der Schritt $K+1$ im Algorithmus schon startet, während der Schritt $K$ noch läuft. Dabei kann natürlich nur Information verwendet werden, die von Schritt $K$ nicht mehr verändert wird. Die meiste Arbeit steckt jedoch in der verschachtelten for-Schleife welche vollständig parallel bearbeitet werden kann. Die Stabilität des Algorithmus (insbesondere die Invertierbarkeit der Diagonablöcke $\tilde{A}(K,K)$) muss mit geeigneter Wahl der Blöcke und der Pivotisierung sichergestellt werden.

Da bei jeder numerischen Berechnung bedingt durch die Rechengenauigkeit von Computern Fehler auftreten gilt für das Residuum der numerischen Lösung $A\mathbf{x} - \mathbf{b} \neq 0$. Eine nachträgliche Verbesserung der Lösung kann durch iterative Verfahren wie z.B. mit Splitting-Verfahren, erreicht werden.

| | | $F$ | | $S_{\text{SIMPLEC}}$ | |
|---|---|---|---|---|---|
| | | Konstr. | Lsg. | Konstr. | Lsg. |
| 0,03M | 1 CPU | 1,3s | 0,12s | 2,5s | 0,22s |
| 0,13M | 4 CPUs | 11s | 0,5s | 32s | 0,19s |
| 0,5M | 16 CPUs | 82s | 2,4s | 203s | 3,6s |
| 2,0M | 64 CPUs | - | | - | |
| 4,0M | 128 CPUs | - | | - | |

Tabelle 6.1: Skalier- und Parallelisierbarkeit der LU-Zerlegung

## 6.2 Benchmarks der LU-Zerlegung

Die im vorherigen Abschnitt beschriebene parallele LU-Zerlegung ist in dem Paket SuperLU_dist [17] implementiert, welches auch für die folgenden Benchmarks eingesetzt wird.

Die Tabelle 6.1 zeigt die Lösungszeiten für Probleme mit $F$ und $S_{\text{SIMPLEC}}$ bei verschiedene Problemdimensionen. Als Benchmark dient das 3D-Problem aus Kapitel 5 mit $0,5M$ Unbekannten, CFL $= 0,1$ und $\nu = 10^{-4}$. Dabei beschreibt die Spalte "Konstr." die Zeit für die Konstruktion der LU-Zerlegung und die Spalte "Lsg." die (durchschnittliche) Zeit zum Lösen des linearen Problems. Für eine zuverlässige Aussage wurden die Lösungszeiten über mehrere Lösungsvorgänge gemittelt. Man beachte, dass mit der Dimension auch die Anzahl der verwendeten CPUs ansteigt. Sie ist gerade so gewählt, dass das Verhältnis zwischen Problemdimension und Anzahl der CPUs konstant ist. Wäre die LU-Zerlegung gut skalier- und parallelisierbar, dürfte nur eine leichte Erhöhung der Ausführungszeit bei steigender Problemdimension feststellbar sein. Stattdessen ist eine starke Abhängigkeit erkennbar. Bei höheren Dimensionen steigt aber nicht nur die Lösungszeit, sondern auch der Speicheraufwand für die Zerlegung rasant an. Für Problemdimensionen $\geq 2,0$M benötigt SuperLU_dist schon nach kurzer Zeit allen verfügbaren Arbeitsspeicher. Der Grund ist die Struktur der Matrizen $F$ und $S_{\text{SIMPLEC}}$ im 3D-Fall. Sie führt dazu, dass die LU-Zerlegung eine volle Matrix ist, was bei diesen Dimensionen nicht mehr speicherbar ist.

Dennoch ist das Verfahren für kleine Dimensionen sehr effizient. Ist die LU-Zerlegung einmal konstruiert, kann sie gespeichert und für alle folgenden Probleme verwendet werden. Insbesondere lohnt sich daher der Einsatz, falls viele Probleme mit der gleichen Matrix, aber unterschiedlichen rechten Seiten berechnet werden müssen.

Besser als in 3D funktioniert die LU-Zerlegung im 2D-Fall. Der Grund ist, dass in diesem Fall $F$ und $S_{\text{SIMPLEC}}$ bei einer geeigneten Zeilen- und Spaltensortierung Diagonalmatrizen mit nahe der Diagonale liegenden Seitenbändern sind. Diese Struktur kann von der LU-Zerlegung genutzt werden, so dass die Dünnbesetztheit in der Zerlegung erhalten bleibt. Damit arbeitet der Algorithmus deutlich effizienter und kann auch noch für größere Dimensionen gut verwendet werden.

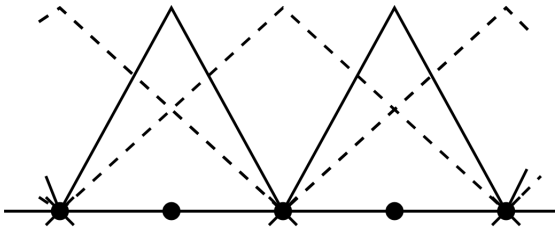

Abbildung 6.1: Basisfunktionen für grobes Gitter $S_g$ (gestrichelt) und Korrekturraum $B$ für das feine Gitter (durchgezogen). Quelle: [4]

## 6.3 Mehrgitterverfahren

Mehrgitterverfahren sind Lösungsverfahren, die speziell für partielle Differentialgleichungen entwickelt sind. Sie sind gut parallelisierbar und in vielen Fällen wird eine Konvergenzrate erreicht, die unabhängig von der Problemdimension ist. Damit gehören Mehrgitterverfahren zu den schnellsten Verfahren dieser Problemklasse.

Betrachtet man Matrizen von diskretisierten partiellen Differentialgleichungen, so stellt man fest, dass deren Eigenwerte sich häufig in hoch- und niederfrequente Bereiche trennen lassen. Viele klassische Iterationsverfahren verkleinern den Fehler im hochfrequenten Anteil schon nach wenigen Iterationen, während der niederfrequente Anteil nur langsam abnimmt. Diese Eigenschaft nutzt das Mehrgitterverfahren, indem es mit einer Hierarchie von Gittern arbeitet. Das feinste davon entspricht dem Gitter, auf dem die partielle Differentialgleichung diskretisiert wurde. Für die Lösung des Problems wird zunächst der hochfrequente Fehler einer Initiallösung mit einem klassischen Iterationsverfahren verkleinert. Anschließend wird das Residuum auf das gröbere Gitter übertragen. Dort wird dann der niederfrequente Fehler berechnet. Die beiden Fehlerterme ergeben schließlich die notwendige Korrektur für die Initiallösung.

Im folgenden wird nun zunächst das 2-Gitterverfahren vorgestellt und dieses dann auf Mehrgitterverfahren verallgemeinern. Als Basis dient ein grobes und ein feines Gitter mit zugehörigen Finite-Elemente-Räumen $S_g$ und $S_f$. Die Gitter seien so gewählt, das $S_g \subset S_f$ gilt, so dass ein Korrekturraum $B$ und eine Zerlegung $S_f = S_g + B$ existiert. Abbildung 6.1 zeigt ein eindimensionales Beispiel mit linearen Basisfunktionen.

Um eine Funktion auf $S_f$ in die Komponenten von $S_g$ und $B$ aufteilen zu können, werden zwei Gittertransferoperatoren benötigt: Der Prolongationsoperator $P : S^g \to S^f$ und der Restriktionsoperator $R : S^f \to S^g$. Da $S^g \subset S^f$ gilt, ist die kanonische Einbettung als Prolongationsoperators eine natürliche Wahl:

$$P(v_g) = v_g \qquad \forall v_g \in S^g$$

Mit der Finite-Elemente-Basis zu $S_f$ und $S_g$ lässt sich der Operator $P$ als Matrix schreiben, die nun ebenfalls $P$ genannt wird. Als Restriktionsoperator bietet sich der zu $P$ transponierte Operator $R := P^T$ an.

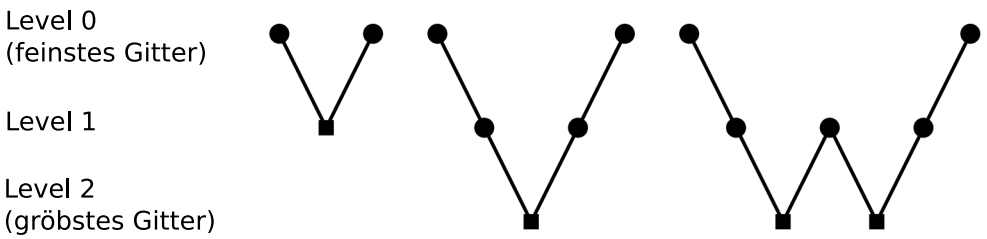


Abbildung 6.2: Beispiel von Mehrgitter-Zyklen von links nach rechts: 2-Gitter, 3-Gitter V-Zyklus und 3-Gitter W-Zyklus.

Mit dieser Vorarbeit lässt sich nun das 2-Gitterverfahren beschreiben. Sei also $A\mathbf{x} = \mathbf{b}$ ein lineares System aus einer Finite-Elemente-Diskretisierung. Der algebraische Fehler eines Startvektors $\mathbf{x}^0$ ist $\mathbf{e}^0 = \mathbf{x} - \mathbf{x}^0$ und kann geschrieben werden als $\mathbf{e}^0 = P\mathbf{e}^0_{S^g} + \mathbf{e}^0_B$ mit dem Fehleranteil $\mathbf{e}^0_{S^g}$ aus $S^g$ und der Feingitter-Korrektur $\mathbf{e}^0_B$. Die zentrale Idee ist nun, ein iteratives Lösungsverfahren - das Glättungsverfahren - zu verwenden, um den Fehleranteil $\mathbf{e}_B$ nach wenigen Schritten numerisch verschwinden zu lassen, so dass nach $i$ Iterationsschritten $\mathbf{e}^i \approx P\mathbf{e}^i_{S^g}$ gilt. Viele einfache Iterationsverfahren, wie zum Beispiel Splitting-Verfahren, haben diese Eigenschaft und können als Glättungsverfahren verwendet werden. Im Idealfall ist $\mathbf{e}^i = P\mathbf{e}^i_{S^g}$, das heißt der Fehler kann auf das grobe Gitter projiziert werden, ohne Informationen zu verlieren. Damit bietet es sich an, das ganze lineare Problem auf das grobe Gitter zu übertragen, um dort den Restfehler zu bestimmen. Man erhält die sogenannte Grobgitterkorrektur:

$$RAP\bar{\mathbf{e}} = R(\mathbf{b} - A\mathbf{x}^i)$$

Der Fehlervektor $\bar{\mathbf{e}} \in S_g$ kann schließlich mit dem Prolongationsoperator auf das feine Gitter zurück übertragen und als Korrektur verwendet werden. Die Lösung des 2-Gitterverfahrens lautet dann also:

$$\mathbf{x}^i + P\bar{\mathbf{e}}$$

Hebt man die Beschränkung von 2 Gittern auf, erhält man das Mehrgitterverfahren. Es gibt dann verschiedene Möglichkeiten, die Glättung mit dem Wechseln zwischen den einzelnen Gitterebenen (oder auch Level) zu verbinden. Die bekanntesten sind der V- und W-Zyklus, wie sie in Abbildung 6.2 skizziert sind. Beim Durchlaufen eines Zyklus folgt der Algorithmus dabei dem Graphen von links nach rechts, so dass beispielsweise der 3-Gitter V-Zyklus vom feinsten Gitter bis zum gröbsten Gitter und anschließend wieder zum feinsten Gitter wechselt.

Der Ablauf des Mehrgitterverfahrens mit V-Zyklus und $n$ Level ist in Algorithmus 2 beschrieben. Dabei sind $R_k$ und $P_k$ die Prolongations- und Restriktionsoperatoren zwischen Level $k$ und Level $k+1$ und $S_k$ ein Lösungsverfahren auf dem Gitter $k$. Häufig ist die Problemdimension auf dem gröbsten Gitter so klein, dass als $S_n$ ein direktes Lösungsverfahren verwendet werden kann. Man bezeichnet den Löser daher

**Algorithmus 2** : Mehrgitterverfahren $\mathbf{x} \leftarrow$ V-Zyklus($A_k$, $\mathbf{b}$, $\mathbf{x}$, $k$)

```
x = S_k(A_k, b, x);
if k ≠ n then
    r = R_k(b − A_k x);
    A_{k+1} = R_k A_k P_k;
    ē = 0;
    ē = V-Zyklus(A_{k+1}, r, ē, k + 1);
    x = x + P_k ē;
```

auch als Grobgitterlöser. Für die restlichen Gitter wählt man als $S_k$ ein geeignetes Glättungsverfahren.

Dieses (klassische) Mehrgitterverfahren hat allerdings den entscheidenden Nachteil, dass bei der Anwendung die Gitter für die Level $> 1$ vorhanden sein müssen, um die Prolongations- und Restriktionsoperatoren zu konstruieren. Wünschenswert wäre dagegen ein "Black Box"-Algorithmus, der nur die Informationen des linearen Problems benötigt. Diesen Ansatz verfolgt das algebraische Mehrgitterverfahren (AMG). Im Folgenden wird das AMG basierend auf "Smoothed Aggregation" vorgestellt [30, 35]. Um den Prolongationsoperator $P_k$ zu erstellen wird zunächst ein Graph aus der Matrix $A_k$ konstruiert. In diesem entspricht jeder Knoten einer Zeile aus $A_k$ und zwei Knoten $i$ und $j$ sind genau dann mit einer Kante verbunden, falls der $(i, j)$- oder $(j, i)$-te Eintrag in $A_k$ nicht Null ist. Dieser Graph kann nun "vergröbert" werden: Zunächst wird die Menge der Knoten in Aggregate partitioniert. Jedes Aggregate entspricht einem Knoten auf dem groben Graphen. Zwei Knoten auf dem groben Graphen sind genau dann mit einer Kante verbunden, wenn es eine mindestens eine Kante zwischen den zugehörigen Aggregaten gibt.

Mit diesem groben Graph definiert man nun einen Prolongationsoperator:

$$\tilde{P}_k(i,j) = \begin{cases} 1, & \text{wenn Knoten } i \text{ ein Element des Aggregats } j \text{ ist.} \\ 0, & \text{sonst.} \end{cases}$$

Obwohl diese Interpolation als Prolongationsoperator verwendet werden kann, erhält man eine robustere Variante durch eine nachträgliche Glättung. Dieses sogenannte "Smoothed Aggregation" erweitert $\tilde{P}_k$ durch einen Glättungsoperator $\tilde{S}_k$ und führt zu $P_k = \tilde{S}_k \tilde{P}_k$. In den folgenden Anwendungen ist $S_k$ ein gedämpfter Jacobi Glätter, siehe [36]. Der Restriktionsoperator wird wie schon beim klassischen Mehrgitterverfahren durch die Transponierte von $P$ definiert: $R := P^T$.

## 6.4 Benchmarks der Mehrgitterverfahren

Da die Teilprobleme $F$ und $S_{\text{SIMPLEC}}$ aus diskretisierten partiellen Differentialgleichungen stammen, ist die Lösung mit dem algebraischen Mehrgitterverfahren erfolgversprechend. Um zufriedenstellende Ergebnisse zu erhalten, ist jedoch eine sehr

| | $F$ | $S_{\text{SIMPLEC}}$ |
|---|---|---|
| Krylov-Unterraum-Verfahren | BiCGStab | BiCGStab |
| Level | 2 | 3 |
| Zyklus | V | V |
| Aggregationsverfahren | METIS | Uncoupled-MIS |
| Glättungsverfahren | Gauß-Seidel | Gauß-Seidel |
| Grobgitterlöser | LU-Zerlegung | LU-Zerlegung |

Tabelle 6.2: AMG Grundeinstellung für F und $S_{\text{SIMPLEC}}$

sorgfältige Parameterwahl und genügend Wissen über das zu lösende Problem notwendig. Im Gegensatz dazu sind Krylov-Unterraum-Verfahren sehr robust, ohne Vorkonditionierer aber ineffizient. Ein Kombination beider, das heißt, ein mit AMG vorkonditioniertes Krylov-Unterraum-Verfahren ist damit die richtige Wahl. Trotzdem ist eine Parameteroptimierung notwendig damit das Verfahren effizient arbeitet.

Ziel dieses Abschnitts ist es, ein gutes Setup für AMG als Vorkonditionierer zu finden und den Einfluss einzelner Parameter sichtbar zu machen. Als Basis dient dafür eine sorgfältig ausgewählte Grundeinstellung zur Lösung von Problemen mit $F$ und $S_{\text{SIMPLEC}}$, siehe Tabelle 6.2. Diese Einstellungen sind das Ergebnis einer händischen Parameteroptimierung und sind so gewählt, dass eine hohe Lösungsgeschwindigkeit bei guter Skalier- und Parallelisierbarkeit erreicht wird.

Um den Aufwand zu begrenzen, werden diese Parameter der Grundeinstellung nun einzeln variiert und die Auswirkung analysiert. Die zu lösenden Teilprobleme stammen wie schon bei der LU-Zerlegung aus dem 3D Benchmark mit 0,5M Unbekannten, CFL $= 0,1$ und $\nu = 10^{-4}$. Die Teilprobleme werden auf eine Genauigkeit von $10^{-3}$ berechnet. Dabei werden folgende Werte gemessen:

1. **Konstr.**: Zeit zur Konstruktion des algebraischen Mehrgitterverfahrens. Dazu gehört die Erstellung der Gitterhierarchie und der Prolongations- und Restriktionsoperatoren sowie die Initialisierung der Glättungsverfahren und des Grobgitterlösers. Letzteres beinhaltet also insbesondere die LU-Zerlegung auf dem gröbsten Gitter, falls diese als Grobgitterlöser verwendet wird.

2. **Lsg.**: Zeit zur Lösung des Gleichungssystems.

3. **Iter.**: Notwendige Iterationen des mit AMG vorkonditionierten Krylov-Unterraum-Verfahrens.

Für eine zuverlässige Aussage werden die Zeiten im Punkt 2 und 3 über mehrere Lösungsvorgänge gemittelt. Für die folgenden Tests wird der Algorithmus seriell (d.h. auf einer CPU) ausgeführt. Die Lösungszeiten mit den Grundeinstellungen aus Tabelle 6.2 zeigt Tabelle 6.3. In den nächsten Abschnitten werden die Parameter der Grundeinstellungen betrachtet und die Auswirkungen bei einer Änderung untersucht.

| $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|
| Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| 18s | 2,8s | 2,0 | 8,1s | 2,2s | 3,9 |

Tabelle 6.3: AMG Zeiten mit den Grundeinstellungen

### 6.4.1 Krylov-Unterraum-Verfahren

Es wurden drei iterative Lösungsverfahren getestet: BiCGStab, GMRES und GMRESR. Da für den Glättungsschritt bei AMG ein iteratives Verfahren verwendet wird, kann der AMG Vorkonditionierer nicht als fester Operator geschrieben werden. Prinzipiell eignet sich daher der GMRESR Algorithmus am besten, da er variable Vorkonditionierer erlaubt. Trotzdem bleibt die Grundstruktur des Vorkonditionierers erhalten und auch GMRES und BiCGStab liefern gute Ergebnisse:

| | $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|---|
| | Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| BiCGStab | 18s | 2,8s | 2,0 | 8,1s | 2,2s | 3,9 |
| GMRES | 18s | 2,7s | 3,0 | 8,1s | 2,0s | 6,2 |
| GMRESR | 18s | 2,4s | 3,0 | 8,1s | 2,0s | 5,9 |

Da BiCGStab für beide Operatoren sehr gute Iterationszahlen aufzeigt, wird er in den Grundeinstellungen verwendet.

### 6.4.2 Level

Die Anzahl der notwendigen Level hängt stark von der verwendeten Aggregation ab (siehe Abschnitt 6.4.3). Wird nur eine geringe Vergröberung pro Level erreicht, werden mehr Level benötigt um die gleiche Problemdimension auf dem gröbsten Gitter zu erhalten. Da jedes Level zusätzlichen Initialisierungsaufwand verursacht und die Initialisierung zudem schlecht parallelisierbar ist, beschränken wir uns hier auf 2 oder 3 Level.

| | $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|---|
| | Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| 2-Level | 18s | 2,8s | 2,0 | 8,5s | 2,1s | 3,7 |
| 3-Level | 25s | 2,8s | 2,0 | 8,1s | 2,0s | 3,9 |

Tatsächlich zeigen die Ergebnisse, dass der AMG-Vorkonditionierer schon mit 2 Level sehr gut funktioniert. Sowohl bei $F$ als auch bei $S_{\text{SIMPLEC}}$ werden nur wenige Krylov-Iterationen benötigt um die Toleranzgenauigkeit zu erreichen. Man beachte, dass sich die Aggregation bei $F$ für das 2-Gitterverfahren von dem 3-Gitterverfahren unterscheidet. Näheres siehe im nächsten Abschnitt.

### 6.4.3 Aggregation

Es werden zwei Aggregationsverfahren betrachtet, die in dem Trilinos Paket ML implementiert sind [30, 13]:

- **Uncoupled-MIS**: Die Vergröberungsrate kann nicht festgelegt werden und liegt bei etwa zwischen 10 und 50 Knoten pro Aggregat.

- **METIS** verwendet einen Graph-Partitionierungsalgorithmus. Die Anzahl der Knoten per Aggregat kann angegeben werden. Sie wurde so gewählt, dass bei einem 2-Level AMG der Graph auf jedem Prozessor auf einen einzigen Knoten reduziert wird. Bei einem 3-Level AMG werden jeweils 512 Knoten zusammengefasst.

Damit lassen sich die Problemgrößen auf den Level angeben. Für $F$ wird ein 2-Gitterverfahren zusammen mit der METIS Aggregation verwendet. Wird auf einem Prozessor gerechnet, besteht das grobe Level aus lediglich einem einzigen Knoten. Anders verhält es sich bei $S_{\text{SIMPLEC}}$. Die Matrix hat eine Dimension von etwa 130.000. Uncoupled-MIS reduziert diese für das erste Level auf etwa 2.200 und auf dem gröbsten Level auf 52.

| | $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|---|
| | Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| METIS | 18s | 2,8s | 2,0 | 12,9s | 6,6s | 12 |
| Uncoupled-MIS | 20s | 2,9s | 2,0 | 8,1s | 2,2s | 3,9 |

Die Tabelle zeigt die Auswirkung der Aggregationsverfahren auf die Testprobleme. Da $F$ die dreifache Dimension von $S_{\text{SIMPLEC}}$ hat, ist der Einsatz von METIS sinnvoll, um eine ausreichend hohe Vergröberungsrate zu erreichen. Mit Uncoupled-MIS und größeren Testproblemen würde die Dimension auf dem gröbsten Level ansteigen, und der Grobgitterlöser nicht mehr effizient arbeiten.

Prinzipiell besteht die gleiche Problematik auch für $S_{\text{SIMPLEC}}$. Da hier aber niedrigere Dimensionen auftreten, ist der Einsatz von Uncoupled-MIS unproblematischer. Insbesondere wird aus der Tabelle deutlich, dass eine zu schnelle Vergröberung eine negative Auswirkung auf die Effektivität von AMG als Vorkonditionierer hat, was als hohe Iterationszahlen des Krylov-Unterraum-Verfahrens sichtbar wird.

### 6.4.4 Zyklus

In der Einleitung der Mehrgitterverfahren wurden der V- und W-Zyklus vorgestellt. Da der W-Zyklus nur für drei oder mehr Level definiert ist und die Grundeinstellung von $F$ ein 2-Gitterverfahren ist, wurde nur $S_{\text{SIMPLEC}}$ getestet.

Man erhält sehr ähnliche Ergebnisse, weshalb der einfachere V-Zyklus in den Grundeinstellungen gewählt wurde.

| | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|
| | Konstr. | Lsg. | Iter. |
| V Zyklus | 8,2s | 2,2s | 3,9 |
| W Zyklus | 8,1s | 2,1s | 3,8 |

### 6.4.5 Glättungsverfahren

Das Glättungsverfahren hat die Aufgabe den hochfrequenten Fehler schnell herauszudämpfen, so dass auf dem nächsten Level weitergerechnet werden kann. Zu den wichtigsten Glättungsverfahren gehören die Jacobi- und Gauß-Seidel-Verfahren, die unvollständige LU-Zerlegung (ILU) und die Chebyshev Methode. Man erhält folgendes Ergebnis:

| | $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|---|
| | Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| Jacobi | | | NC | 8,0s | 2,2s | 4,9 |
| Gauß-Seidel | 18s | 2,8s | 2,0 | 8,1s | 2,2s | 3,9 |
| ILU | 416s | 1,9s | 1,0 | 62s | 1,6s | 2,1 |
| Chebyshev | 18s | 3,1s | 3,9 | 8,0s | 1,7s | 4,9 |

Da mit ILU eine unvollständige LU-Zerlegung durchgeführt werden muss, ist die Konstruktion verhältnismäßig zeitaufwändig. Dafür sind die Lösungszeiten niedriger. Um die Geschwindigkeit der ILU-Zerlegung zu steigern, wurden mehrere Varianten entwickelt, z.B. ILUT. Da sie aber in der Regel benutzerdefinierte Parameter benötigen, soll hier nicht näher darauf eingegangen werden.

Auch der Jacobi Glätter benötigt einen problemspezifischen Dämpfungsparameter, um effizient zu arbeiten. In diesem Fall wurde $\frac{4}{5}$ gewählt. Die optimale Dämpfung hängt jedoch im Allgemeinen von der zugrundeliegenden partiellen Differentialgleichung ab. Daher könnte eine günstigere Wahl zu einer Verbesserung bei der Lösung mit F führen.

Die Gauß-Seidel und Chebyshev Glättungsverfahren lieferen ähnlich gute Ergebnisse und kommen ohne benutzerdefinierte Parameter aus. Es sollte allerdings erwähnt werden, dass Gauß-Seidel im Gegensatz zum Jacobi-Verfahren nicht einfach parallelisierbar ist. Daher wird in parallelen Umgebungen eine Variante verwendet: Die Gauß-Seidel Iteration wird unabhängig auf jedem Prozessor durchführt wird, wobei mit den Informationen der anderen Prozessoren von der vorherigen Iteration gerechnet wird.

Insgesamt führt der Gauß-Seidel Glätter auf gute Iterationszahlen und benötigt keine benutzerspezifischen Parameter, weshalb dieser in den Grundeinstellungen verwendet wird.

### 6.4.6 Grobgitterlöser

Auf dem gröbsten Gitter wird häufig ein spezielles Lösungsverfahren verwendet, dass sich von den Glättungsverfahren unterscheidet. Drei verschiedene Optionen wurden als Grobgitterlöser untersucht: die LU-Zerlegung, die Jacobi- und das Gauß-Seidel-Verfahren.

| | $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|---|
| | Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| LU | 18s | 2,7s | 2,0 | 8,1s | 2,2s | 3,9 |
| Jacobi | 18s | 2,7s | 2,0 | 8,1s | 6,1s | 12 |
| Gauß-Seidel | 18s | 2,7s | 2,0 | 8,1s | 4,9s | 9,3 |

Im Fall $F$ ist es nicht verwunderlich, dass das Ergebnis für alle 3 Lösungsverfahren gleich sind: Aufgrund der METIS-Aggregation besteht das gröbste Gitter nur aus einem Knoten und damit hat auch das zu lösende Problem die Dimension 1. Bei einem 1-dimensionalen Problem sind jedoch LU-Zerlegung, Jacobi und Gauß-Seidel äquivalent.

Anders ist es im Fall von $S_{\text{SIMPLEC}}$. Das Lösen der $52 \times 52$ großen Matrix mit der LU-Zerlegung ist schneller als mit iterativen Verfahren. Allerdings hat die Matrix auf dem gröbsten Level typischerweise wenige Null-Einträge und bei feineren Gittern wird die LU-Zerlegung schnell ineffektiv. In diesem Fall werden die iterativen Löser attraktiv. Zudem sind letztere besser skalierbar und versprechen eine bessere Parallelität.

Da die LU-Zerlegung sehr gute Ergebnisse liefert, wird sie auch in den Grundeinstellungen verwendet. Im nächsten Abschnitt werden allerdings auch sehr hochdimensionale Probleme gelöst, bei denen stattdessen der Gauß-Seidel Algorithmus eingesetzt wird.

### 6.4.7 Skalierbarkeit und Parallelisierbarkeit

Grundvoraussetzung für eine gute Parallelperformance ist, dass stets alle Prozessoren beschäftigt sind. Gerade bei vielen Prozessoren kann es allerdings passieren, dass auf den gröberen Level die Aufgaben schlecht verteilt sind. Die Folge ist, dass viele Prozessoren auf einige wenige warten müssen, um die Berechnung fortsetzen zu können. Aus diesem Grund ist eine dynamische Lastverteilung wichtig. Das Trilinos Paket ML bietet eine automatische Lastverteilung mit Zoltan [37] an, die in den folgenden Benchmarks aktiviert ist.

Tabelle 6.4 zeigt die Skalier- und Parallelisierbarkeit der optimierten Grundeinstellungen. Dabei ist die Anzahl der CPUs so gewählt, dass das Verhältnis der Problemdimension und der Anzahl der CPUs konstant ist. Da das Problem mit 4.0M auf dem gröbsten Gitter noch zu hochdimensional ist, um eine LU-Zerlegung zu berechnen, werden stattdessen einige Gauß-Seidel Iterationen durchgeführt. Insgesamt kann die Skalier- und Parallelisierbarkeit der Mehrgitterverfahren als sehr gut eingestuft werden. Sowohl die Initialisierungs- als auch die Lösungszeiten zeigen nur eine

| | | $F$ | | | $S_{\text{SIMPLEC}}$ | | |
|---|---|---|---|---|---|---|---|
| | | Konstr. | Lsg. | Iter. | Konstr. | Lsg. | Iter. |
| 0,03M | 1 CPU | 0,75s | 0,11s | 2,0 | 0,25s | 0,062s | 3,9 |
| 0,13M | 4 CPUs | 1,4s | 0,23s | 2,0 | 0,70s | 0,21s | 3,9 |
| 0,5M | 16 CPUs | 1,5s | 0,28s | 2,1 | 0,87s | 0,30s | 4,7 |
| 2,0M | 64 CPUs | 3,5s | 0,67s | 2,0 | 2,3s | 0,81s | 4,6 |
| 4,0M | 128 CPUs | 4,4s | 0,82s | 2,1 | 2,3s | 1,8s | 9,0 |

Tabelle 6.4: Skalier- und Parallelisierbarkeit von AMG

leichte Abhängigkeit von der Problemdimension. Bemerkenswert ist auch die hervorragende Qualität des AMG-Vorkonditionierers, der fast dimensionsunabhängige und sehr niedrige Iterationszahlen liefert. Auffällig ist der Zeitsprung zwischen 16 und 64 Prozessoren. Dieser Zeitsprung hat folgenden technischen Hintergrund: Jeder Rechner im Cluster besitzt 32 Prozessoren. Bei Verwendung von mehr als 32 CPUs muss daher die Kommunikation über das verhältnismäßig langsame Netzwerk geschehen.

# Kapitel 7

# Benchmark des Gesamtsystems

Nachdem mit Newton-GMRESR-SIMPLEC bzw. Newton-GMRESR-SIMPLER gute Lösungsverfahren ausgewählt wurden und die dabei auftretenden Teilprobleme mit AMG effizient gelöst werden können, kann der Zeitbedarf beim Lösen der instationären Navier-Stokes-Gleichungen betrachtet werden.

Tabelle 7.1 führt die einzelnen Programmabschnitte und deren Zeitbedarf auf (siehe auch Abbildung 3.1 auf Seite 21). Man beachte, dass die Matrix $C$ außerhalb der Newtonschleife assembliert wird, da sie durch die Newtonkorrektur nicht verändert wird. Als Lösungsverfahren des linearen Systems wurde GMRESR-SIMPLEC verwendet.

| | Zeit bei 1 CPU | Zeit bei 16 CPUs | Faktor |
|---|---|---|---|
| Einlesen des Gitters: | 99s | 23s | 4 |
| Aufstellen der Gleichungen: | 13s | 3,4s | 4 |
| **Zeitschleife** | | | |
| Assemblierung von $C$: | 32s | 5,2s | 6 |
| **Newtonschleife** | | | |
| Assemblierung von $F$, $B^T$, $B$: | 503s | 77s | 7 |
| Vorkonditionierer erstellen: | 125s | 35s | 4 |
| Lösen des LGS: | 44s | 4,9s | 9 |
| Newtonkorrektur: | 4,0s | 0,8s | 5 |
| **Ende der Newtonschleife** | | | |
| **Ende der Zeitschleife** | | | |

Tabelle 7.1: Zeitmessung des Gesamtsystems

Auffällig bei der Messung ist der hohe Zeitbedarf für die Assemblierung der Matrizen. Für eine gute Performance des Gesamtsystems ist es also wichtig, die Toleranzbedingung mit so wenig Newton-Iterationen wie möglich zu erreichen. Das wiederum bedeutet, dass es nicht von Vorteil ist, die linearen Gleichungssystem mit der Blockmatrix $(F, B^T; B, C)$ in der Newtonschleife aus Zeitgründen nur ungenau zu lösen,

| | Zeit bei 1 CPU |
|---|---|
| Initialisierung von $F^{-1}$: | 18s |
| Explizite Berechnung von $S_{\text{SIMPLEC}}$: | 98s |
| Initialisierung von $S^{-1}_{\text{SIMPLEC}}$: | 8,7s |

Tabelle 7.2: Initialisierungszeit des SIMPLEC-Vorkonditionierers (1 CPU)

da dann eventuell mehr Newton-Iterationen notwendig sind. Es hat sich gezeigt, dass eine Fehlertoleranz von $10^{-5}$ für diese linearen Probleme eine günstige Wahl ist.

Eine andere Situation liegt beim Lösen der Teilprobleme $F$ und $S_{\text{SIMPLEC}}$ vor, die bei der Anwendung des Vorkonditionierers berechnet werden müssen. Werden diese nur ungenau gelöst, kann der SIMPLEC-Vorkonditionierer nicht als konstanter Operator beschrieben werden. Da aber GMRESR als Lösungsverfahren verwendet wird und dieser einen variablen Vorkonditionierer erlaubt, hat das wenig Einfluss auf die notwendigen GMRESR-Iterationen. Die Praxis hat gezeigt, dass eine Toleranz von $10^{-3}$ für die Teilprobleme gut ausreicht, ohne die Zahl der GMRESR-Iterationen merklich zu erhöhen.

Den zweitgrößten Zeitaufwand in Tabelle 7.1 benötigt die Erstellung des Vorkonditionierers. Die genaue Zeitaufteilung dieses Schrittes zeigt Tabelle 7.2. Offensichtlich wird die meiste Zeit für die expliziten Berechnung des approximativen Schurkomplements $S_{\text{SIMPLEC}} = B(\sum |F|)^{-1}B^T + C$ benötigt. Die restliche Zeit vergeht bei der Initialisierung der Mehrgitterverfahren für $F$ und $S_{\text{SIMPLEC}}$. Zunächst ist nicht klar, wieso die explizite Berechnung von $S_{\text{SIMPLEC}}$ überhaupt notwendig ist: das Lösen eines linearen Systems mit $S_{\text{SIMPLEC}}$ durch ein iteratives Lösungsverfahren wie BiCGStab benötigt lediglich Matrix-Vektor Operationen, die auch ohne die explizite Form von $S_{\text{SIMPLEC}}$ durchgeführt werden können. Das Problem ist die AMG-Vorkonditionierung: Beim Erstellen der AMG Hierarchie (siehe Abschnitt 6.3) wird ein Graph erstellt, der von den Positionen der Nicht-Nulleinträge in der Matrix abhängt. Bei erweiterten Hierarchiemethoden werden zusätzlich kleine Matrixeinträge ignoriert, womit sogar der Wert jedes Matrixeintrags bekannt sein muss[1].

Tabelle 7.1 listet zusätzlich die Zeiten beim Einsatz von 16 CPUs auf, sowie den Beschleunigungsfaktor gegenüber der Lösung mit einer CPU. Wie man sieht, kann jeder Schritt die zusätzlichen Ressourcen relativ gut nutzen. Besonders gut schneidet das Lösen des linearen Gleichungssystems ab. Schlecht parallelisierbar ist dagegen die Assemblierung der Matrizen und die Konstruktion des Vorkonditionierers.

[1] Eine iterative Lösung ohne AMG-Vorkonditionierung wurde testweise ebenfalls durchgeführt. Zwar fällt dann die explizite Berechnung von $S_{\text{SIMPLEC}}$ weg, dafür konvergieren die Krylov-Verfahren nur sehr schlecht oder gar nicht.

# Kapitel 8

# Zusammenfassung

In dieser Arbeit wurden verschiedene Lösungsstrategien für die stabilisierten instationären Navier-Stokes-Gleichungen untersucht. Ein Vergleich dieser Strategien ergab, dass die Kombination eines Newton-Verfahrens für das nichtlineare Problem zusammen mit einem vorkonditionierten GMRESR-Verfahren zur Berechnung der linearen Gleichungssysteme besonders günstig ist. Insgesamt vier Block-Vorkonditionierer für GMRESR wurden miteinander verglichen. Zwei davon sind klassische Druck-Korrektur-Verfahren, die ursprünglich als iterative Lösungsverfahren entworfen sind, aber auch als Vorkonditionierer gut funktionieren. Die anderen beiden gehören zu den neueren Approximate-Commutator-Vorkonditionierern. Sie verwenden Teile der LDU-Zerlegung der zu lösenden Blockmatrix und unterscheiden sich nur von der Approximation des Schurkomplements. All diese Vorkonditionierer haben gemeinsam, dass bei ihrer Anwendung wiederum lineare Probleme auftreten, die jedoch im Vergleich zu dem Ursprungsproblem deutlich einfacher zu lösen sind. Auch für diese Teilprobleme wurde nach effizienten Lösungsverfahren gesucht.

Die numerischen Tests wurden für ein Bechmarkproblem in 2D und 3D durchgeführt. Da der Einsatz von Supercomputern für viele Problemstellungen notwendig ist, spielte bei der Analyse der Verfahren deren Skalier- und Parallelisierbarkeit eine wichtige Rolle. Das Ergebnis der Untersuchung ergab, dass das Newton-Verfahren zusammen mit GMRESR und SIMPLEC- oder SIMPLER als Vorkonditionierer die beste Lösungsmethode für das Benchmarkproblem ist. Für die auftretenden Teilprobleme hat sich das algebraische Mehrgitterverfahren als besonders geeignet herausgestellt. Damit konnte insgesamt eine Lösungsstrategie entwickelt werden, die weitgehend unabhängig von den gewählten Problemparametern wie Gitterweite, Viskosität und Zeitschrittweite effizient arbeitet.

# Anhang A

# Softwarebeschreibung

Dieser Arbeit liegt eine DVD bei, die die entwickelten Bibliotheken und Benchmarkprogramme enthält. Außerdem ist für jeden durchgeführten Benchmark die komplette Konfiguration gespeichert worden. Das beinhaltet insbesondere den Programmcode und die gewählten Parameter, so dass alle Ergebnisse einfach und vollständig nachvollzogen werden können.

## A.1 Installation der Vorkonditionierer

Für die Installation wird Trilinos in der Version 9.0 [29] benötigt. Bei der Installation muss darauf geachtet werden, dass alle notwendigen Pakete installiert werden. Für die vorgestellten Vorkonditionierer sind das die Pakete Thyra, Epetra, Stratimikos, Anasazi und Meros. Für die Testbeispiele wird zudem noch Sundance, Zoltan, AztecOO und Amesos benötigt. Möchte man mit der LU-Zerlegung arbeiten, sollte zusätzlich die SuperLU_dist Unterstützung mit kompiliert werden.

Die Installation der Vorkonditionierer wurde so einfach wie möglich gestaltet. Die DVD enthält einen Ordner *Trilinos/meros*. Der Inhalt muss in das Trilinos Programmverzeichnis *packages/meros* kopiert werden. Anschließend muss das *./configure* Skript von Trilinos erneut aufgerufen werden und die neuen Bibliotheken mit dem *make* Befehle kompiliert werden. Ein *make install* installiert schließlich die neuen Bibliotheken und diese können daraufhin verwendet werden.

Die Testbeispiele benutzen GMRESR als Krylov-Unterraum-Verfahren. Leider ist dieses Verfahren in Trilinos 9.x nicht aktiviert. Um es zu benutzen zu können, muss die Datei *packages/stratimikos/adapters/aztecoo/src/AztecOOParameterList.cpp* geöffnet werden und die Zeilen

```
1       tuple<std::string>("CG","GMRES","CGS","TFQMR","BiCGStab","LU"),
2       tuple<int>(AZ_cg,AZ_gmres,AZ_cgs,AZ_tfqmr,AZ_bicgstab,AZ_lu),
```

ersetzt werden durch

```
1       tuple<std::string>("CG","GMRES","GMRESR","CGS","TFQMR","BiCGStab","LU"),
2       tuple<int>(AZ_cg,AZ_gmres,AZ_GMRESR,AZ_cgs,AZ_tfqmr,AZ_bicgstab,AZ_lu),
```

Zum Testen der Installation ist ein Minimalbeispiel beigelegt. Dieses befindet sich auf der DVD in dem Ordner *Trilinos/meros_ example*. Es wird die 3D Navier-Stokes-Gleichung mit Newton-GMRESR-SIMPLEC Algorithmus benutzt, kann aber mit wenigen Änderungen auf die anderen Vorkonditionierer umgeschrieben werden. Mehr Beispiele finden sich zudem in dem Benchmark Ordner.

## A.2 Benchmarkprogramme

In dem DVD-Verzeichnis *Benchmarks* sind alle durchgeführten Benchmarks in dieser Arbeit gespeichert. Neben den eigentlichen Programm- und Parameterdateien sind auch die Programmausgaben gesichert. Letztere sind an dem Namen *batch[...].out* widererkennbar.

Für jedes untersuchte Lösungsverfahren X aus Kapitel 5 gibt es zwei Ordner *Benchmark X* und *Benchmark X 3D*, die die 2D bzw. 3D Benchmarks enthalten. Ein Openoffice-Dokument erklärt die genaue Verzeichnisstruktur. Der Ordner *Benchmark Subsolver 3D* enthält die Benchmarks aus Kapitel 6. In Verzeichnis *Benchmark Timing 3D* sind schließlich die Ergebnisse für Kapitel 7 gespeichert. Zusätzlich sind die folgenden Ordner vorhanden:

**Meshes** Enthält die 2D-Gitter $0,03M$, $0,13M$ und $0,5M$, die für die Benchmarks verwendet wurden. Aus Platzgründen konnten die Gitter mit 2 Millionen und 4 Millionen Unbekannten nicht auf die DVD gespeichert werden.

**Meshes 3D** Analog für die 3D-Gitter.

**Startwerte** Hier sind 2D-Lösungen zum Zeitpunkt $t = 2,0$ gespeichert, um sie als Startwerte in das Benchmarkprogramm einlesen zu können.

**Startwerte 3D** Analog für die 3D-Lösungen.

# Literaturverzeichnis


[1] D. Braess. *Finite Elemente.* Springer, 3 edition, 2003.

[2] A. N. Brooks and T. J. R. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Comput. Methods Appl. Mech. Eng.*, pages 199–259, 1990.

[3] H. Elman, V. E. Howle, J. Shadid, D. Silvester, and R. Tuminaro. Least squares preconditioners for stabilized discretizations of the navier-stokes equations. *SIAM Journal on Scientific Computing*, 30(1):290–311, 2007.

[4] H. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solver with applications in incompressible fluid dynamics.* Oxford Science Publications, 2005.

[5] H.C. Elman. Preconditioning for the steady-state navier–stokes equations with low viscosity. *SIAM Journal on Scientific Computing*, 20:1299–1316, 1999.

[6] C. Großmann and H.-G. Roos. *Numerische Behandlung partieller Differentialgleichungen.* Vieweg+Teubner, 3 edition, 2005.

[7] J. Hoffman. Computation of mean drag for bluff body problems using adaptive dns/les. *SIAM Journal on Scientific Computing*, 27(1):184–207, 2005.

[8] J. Hoffman. Adaptive simulation of the subcritical flow past a sphere. *Journal of Fluid Mechanics*, 568:77–88, 2006.

[9] J. Hoffman and C. Johnson. A new approach to computational turbulence modeling. *Comput. Methods Appl. Mech. Engrg.*, 2006.

[10] V. Howle, J. Schroder, and R. Tuminaro. The effect of boundary conditions within pressure convection–diffusion preconditioners. Technical Report SAND2006-4466, Sandia National Laboratories, Livermore, CA, 2006.

[11] C. Johnson J. Hoffman. *Applied Mathematics: Body and Soul Computational Turbulent Incompressible Flow*, volume 4. Springer, Berlin, 2007.

[12] O. A. Karakashian. On a galerkin-lagrange multiplier method for the stationary navier-stokes equations. *SIAM Journal on Numerical Analysis*, 19(5):909–923, 1982.

[13] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical report, In Proceedings of Supercomputing, 1998.

[14] C. Li and C. Vuik. Some results on the eigenvalue analysis of a simpler preconditioned matrix. *Department of Applied Mathematical Analysis, Delft University of Technology*, Report 03-08, 2003.

[15] C. Li and C. Vuik. The GCR-SIMPLE solver and the SIMPLE-type preconditioning for incompressible Navier Stokes equations. In P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, and D. Knörzer, editors, *Proceedings of the $4^{th}$ European Congress on Computational Methods in Applied Sciences and Engineering, Volume II, ECCOMAS 2004*, Jyväskylä, 2004. University of Jyväskylä.

[16] X. S. Li and J. W. Demmel. Making sparse gaussian elimination scalable by static pivoting. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–17, Washington, DC, USA, 1998. IEEE Computer Society.

[17] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.

[18] D. Loghin, A. Wathen, and H. Elman. Preconditioning techniques for newton's method for the incompressible navier–stokes equations. *BIT*, 43:961–974, 2003.

[19] P. R. McHugh and D. A. Knoll. Comparison of standard and matrix-free implementations of several Newton-Krylov solvers. *AIAA Journal*, 32:2394–2400, December 1994.

[20] M. F. Murphy, G. H. Golub, and A. J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM Journal on Scientific Computing*, 21(6):1969–1972, 2000.

[21] S. V. Patankar. A calculation procedure for two-dimensional elliptic situations. *Numerical Heat Transfer*, 4:409–425, December 1981.

[22] M. Polner. *Galerkin Least-Squares Stabilization Operators for the Navier-Stokes Equations : A Unified Approach.* PhD thesis, University of Twente, Enschedeq, 2005.

[23] A. Quarteroni, F. Saleri, and A. Veneziani. Factorization methods for the numerical approximation of Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 188(1-3):505–26, 2000.

[24] H.-G. Roos, M. Stynes, and L. Tobiska. *Numerical Methods for Singularly Perturbed Differential Equations: Convection–Diffusion and Flow Problems*. Springer, Berlin, 1996.

[25] Y. Saad and M. H. Schultz. Gmres: a generalized minimal residual method for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[26] R. Shuttleworth. *Block Preconditioners Based on Approximate Commutators*. PhD thesis, University of Maryland, 2007.

[27] D. Silvester, H. Elman, D. Kay, and A. Wathen. Efficient preconditioning of the linearized navier–stokes equations for incompressible flow. *Journal on Computational and Applied Mathematics*, 128:261–279, 2001.

[28] T. E. Tezduyar. Stabilized finite element formulations for incompressible flow computations. *Adavances in Applied Mechanics*, 28:1–44, 1992.

[29] Trilinos. http://trilinos.sandia.gov/. (besucht am 13.06.2009).

[30] Ray S. Tuminaro. Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 5, Washington, DC, USA, 2000. IEEE Computer Society.

[31] S. Turek and M. Schäfer. Benchmark computations of laminar flow around cylinder. In E.H. Hirschel, editor, *Flow Simulation with High–Performance Computers II*, volume 52 of *Notes on Numerical Fluid Mechanics*, pages 547–566. Vieweg, 1996. co. F. Durst, E. Krause, R. Rannacher.

[32] H. A. van der Vorst. Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[33] H.A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Num. Lin. Alg. Appl.*, 1:369–386, 1994.

[34] J. P. Van Doormaal and G. D. Raithby. Enhancements of the simple method for predicting incompressible fluid flows. *Numerical Heat Transfer, Part A: Applications*, 7(2):147–163, 1984.

[35] P. Vanek, M. Brezina, and J. Mandel. Acceleration of convergence of a two level algorithm by smooth transfer operatos. *Appl. Math.*, 37:S. 265–274, 1992.

[36] P. Vanek, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56:179–196, 1996.

[37] Zoltan: Parallel partitioning, load balancing and data-management services. http://www.cs.sandia.gov/~kddevin/Zoltan_pdf/ug.pdf, 2007. (besucht am 13.06.2009).